

SIEMENS

WinCC

Configuration Manual

Manual Volume 2

This manual is part of the documentation package
with the order number:

6AV6392-1CA05-0AB0

C79000-G8276-C158-01

Release: September 1999

WinCC, SIMATIC, SINEC, STEP are trademarks of Siemens.

The other names used in this manual may be trademarks; their owners' rights may be violated if they are used by third parties for their own purposes.

(The transmission and reproduction of this document, and utilization and disclosure of its contents are not permitted unless expressly authorized.
Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.)

(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 improvements are welcomed.)

© Siemens AG 1994 - 1999 All rights reserved

Technical data subject to change

C79000-G8276-C158
Printed in the Federal Republic of Germany

Siemens Aktiengesellschaft

Table of Contents

1	Starting Up the Samples	1-1
1.1	Downloading the Samples.....	1-2
1.2	Starting Up the Samples (Single-User Projects).....	1-4
2	Tag/Variable Configuration (Project_TagHandling).....	2-1
2.1	Creating, Grouping and Moving Tags	2-2
2.2	Incrementing, Decrementing, Jogging	2-8
2.2.1	Jogging - Set-Point Value Change (example 01).....	2-9
2.2.2	Jogging - Set-Point Value Change via Global Script (example 02).....	2-11
2.2.3	Jogging - Button (example 05)	2-14
2.2.4	Jogging - Changeover Switch (example 06)	2-18
2.2.5	Incrementing and Decrementing (example 01).....	2-20
2.2.6	Incrementing and Decrementing via Global Script (example 02)...	2-24
2.2.7	The remaining Samples of this Topic.....	2-28
2.3	Changing Tag Values via Windows Objects	2-29
2.3.1	Input via a Slider with Direct Connection (example 01)	2-30
2.3.2	Input via a Slider and Tag Connection (example 03).....	2-33
2.3.3	Input via an Option Group (Radio-Button) (example 02)	2-34
2.3.4	Input via a Check-Box (example 04)	2-37
2.4	Bit Processing in Words	2-40
2.4.1	Setting a Bit directly via a Check-Box and Direct Connection (example 06).....	2-41
2.4.2	Selecting a Bit and Changing its Status (example 01).....	2-44
2.4.3	The remaining Samples of this Topic.....	2-48
2.5	Indirect Addressing of Tags.....	2-49
2.5.1	Indirect Addressing via a Direct Connection (example 01)	2-50
2.5.2	Multiplex Display with Indirect Addressing and C-Action (example 02)	2-52
2.5.3	Indirect Addressing with C-Action (example 03)	2-54
2.5.4	The remaining Samples of this Topic.....	2-56
2.6	Simulation of Tags.....	2-57
2.6.1	Simulation of a Triangular Oscillation via a C-Action (example 01)	2-58
2.6.2	Simulation via an External Program (example 02).....	2-61
2.7	Importing / Exporting Tags	2-63
2.8	Using Structure Tags.....	2-65
2.8.1	Controlling a Valve with a Structure Tag (example 01).....	2-66
3	Picture Configuration (Project_CreatePicture)	3-1
3.1	Screen Layout and Picture Change	3-3
3.1.1	Screen Layout	3-4

3.2	Picture Change.....	3-6
3.2.1	Opening a Picture via a Direct Connection and Displaying the Picture Name (example 01).....	3-7
3.2.2	Opening a Picture via the Dynamic Wizard (example 02).....	3-11
3.2.3	Opening a Picture via an Internal Function (example 02).....	3-13
3.2.4	Single Picture Change via the Dynamic Wizard (example 03)	3-14
3.2.5	Single Picture Change via a Direct Connection (example 04)	3-16
3.2.6	Opening a Picture via the Object Name and an Internal Function (05)	3-18
3.2.7	Opening a Picture via the Object Name and a Tag Connection with Display of the Picture Name (example 06)	3-20
3.3	Displaying a Picture Window.....	3-23
3.3.1	Hiding (Deselection) and Displaying (Selection) from outside the Picture Window (example 01)	3-24
3.3.2	Displaying (Selection) from outside and Hiding (Deselection) from within the Picture Window (example 02).....	3-26
3.3.3	Time-Controlled Hiding of a Picture (example 03)	3-28
3.3.4	Displaying a Picture Window while the Right Mouse Button is Pressed (example 04)	3-30
3.3.5	Configuring Information Boxes with the Wizard (example 05)	3-31
3.3.6	Displaying a Dialog for Text Input (example 06)	3-35
3.4	Operator-Control Enable	3-37
3.4.1	Exiting Runtime and System (example 01)	3-38
3.4.2	Operator-Control Enable, Logon with Default Box (example 02)...	3-40
3.4.3	Operator-Control Enable, Logon via a separate Dialog (example 03).....	3-43
3.5	Picture Zoom	3-45
3.5.1	Changing the Picture Geometry between two Sizes (example 01)	3-46
3.5.2	Changing the Picture Geometry Continuously (example 02).....	3-49
3.5.3	Configuring an adjustable Picture Geometry via the Properties Dialog (example 03)	3-51
3.6	Control Windows	3-52
3.6.1	Binary Switching Operation (Two-Step Control) (example 01)	3-53
3.6.2	Binary S-R Switching Operation (Two-Step Control) (example 02)	3-55
3.6.3	Binary Switching Operation with Acknowledgment (example 03) ..	3-57
3.6.4	Automatic Input Check (example 04)	3-59
3.6.5	Enhanced Automatic Input Check (example 05).....	3-61
3.6.6	Multiple Operation (example 06)	3-65
3.7	Dynamization.....	3-69
3.7.1	Color Change (example 01)	3-70
3.7.2	Text Change (example 02).....	3-73
3.7.3	Animation of Movement (example 03).....	3-74
3.7.4	Displaying and Hiding Objects using a Bit Evaluation (example 04).....	3-75
3.7.5	Animation of Movement via a C-Action (example 05)	3-77

3.7.6	Creating Animation of Movement with a Wizard (example 06)	3-79
3.7.7	Color Change via a C-Action (example 06).....	3-81
3.7.8	Animation of Movement via a Status Display (example 07).....	3-83
3.8	Language Switch	3-85
3.8.1	Runtime Language Switch (example 01)	3-86
3.8.2	Dialog Box for the Runtime and Control Center Language Switch (example 02).....	3-87
3.9	Operation without a Mouse	3-88
3.9.1	Operation via TAB Key or Hotkey (example 01)	3-89
3.9.2	Cursor Keyboard (example 02)	3-98
3.9.3	Entering Values, Switching Operations (example 03).....	3-103
3.10	Displaying and Hiding Information.....	3-107
3.10.1	Displaying and Hiding Objects (example 01)	3-108
3.10.2	Date and Time Display (example 02).....	3-110
4	WinCC Editors (Project_WinCCEditors)	4-1
4.1	Tag Logging.....	4-2
4.1.1	Cyclic-Continuous Archiving (ex_3_chapter_01.pdf)	4-3
4.1.2	Cyclic-Selective Archiving (ex_3_chapter_01a.pdf)	4-18
4.1.3	Archiving if Values are Exceeded (ex_3_chapter_01b.pdf)	4-27
4.1.4	User-Defined Table Layout (ex_3_chapter_01c.pdf)	4-40
4.1.5	Archiving Binary Tags (ex_3_chapter_01d.pdf)	4-49
4.1.6	Archiving at Defined Times (ex_3_chapter_01e.pdf)	4-56
4.1.7	Exporting Archives (ex_3_chapter_01f.pdf)	4-62
4.2	Alarm Logging	4-70
4.2.1	Bit Message Procedure (ex_3_chapter_02.pdf).....	4-71
4.2.2	Limit Value Monitoring (ex_3_chapter_02a.pdf).....	4-84
4.2.3	Limit Value Monitoring (Continuation)	4-89
4.2.4	Message Window (ex_3_chapter_02b.pdf).....	4-103
4.2.5	Message Archiving (ex_3_chapter_02c.pdf)	4-108
4.2.6	Group Messages (ex_8_generator_00.pdf)	4-115
4.3	Report Designer	4-122
4.3.1	Picture Documentation (ex_3_chapter_03.pdf).....	4-123
4.3.2	Reporting of the WinCC Explorer (ex_3_chapter_03.pdf).....	4-132
4.3.3	Reporting of Tag Logging CS (ex_3_chapter_03.pdf)	4-135
4.3.4	Printing Out Trend Windows in Runtime (ex_3_chapter_01a.pdf) .	4-137
4.3.5	Printing Out Tables in Runtime (ex_3_chapter_01c.pdf)	4-144
4.3.6	Message Sequence Report (ex_3_chapter_02b.pdf)	4-148
4.3.7	Message Sequence Report on a Line Printer	4-151
4.3.8	Message Archive Report (ex_3_chapter_02c.pdf)	4-153
4.4	OLE Communication with EXCEL.....	4-155
4.4.1	Reading and Writing Tag Values (ex_3_chapter_04.pdf)	4-156
4.5	Additional Configurations in the Samples	4-160

4.5.1	Picture Index.....	4-161
4.5.2	Index.....	4-165
4.5.3	Color Dialogs (ex_3_chapter_01c).....	4-168
4.5.4	Bar Graph Display (ex_3_chapter_01e).....	4-172

Preface

Purpose of the Manual

This manual introduces you to the configuration options available with WinCC by means of the following sections:

- Starting up the Samples
- Tag/Variable Configuration
- Picture Configuration
- WinCC Editors

This manual is available in printed form as well as an electronic online document.

The table of contents or the index will quickly point you to the information desired. The online document also offers an expanded search function.

Requirements for Using this Manual

Basic knowledge of WinCC, for example from the Getting Started manual or through practical experience in the configuration with WinCC.

Additional Support

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

In addition, you can contact our Hotline at the following number:

+49 (911) 895-7000 (Fax -7001)

Information about SIMATIC Products

Constantly updated information about SIMATIC products can be found in the CA01 catalog. This catalog can be accessed at the following Internet address:

<http://www.ad.siemens.de/ca01online/>

In addition, the Siemens Customer Support provides you with current information and downloads. A compilation of frequently asked questions is available at the following Internet address:

http://www.ad.siemens.de/support/html_00/index.shtml


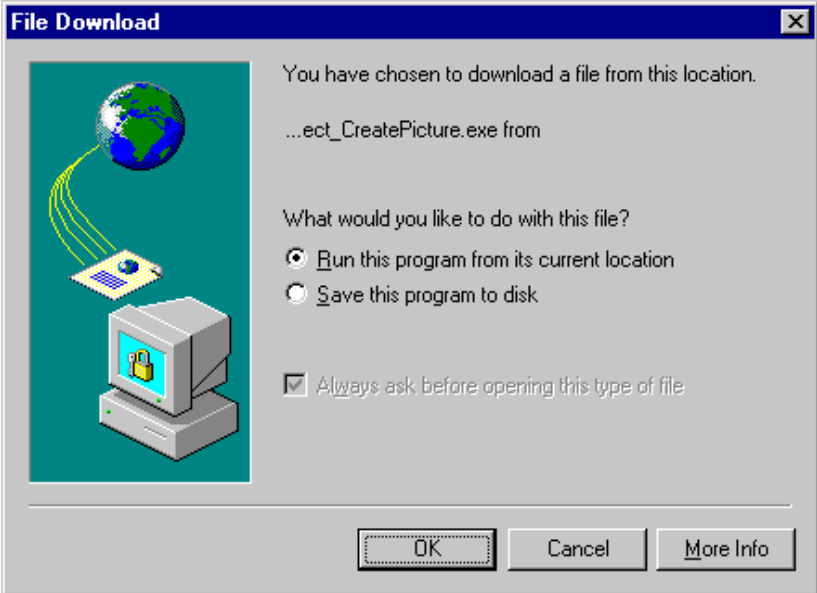
1 Starting Up the Samples



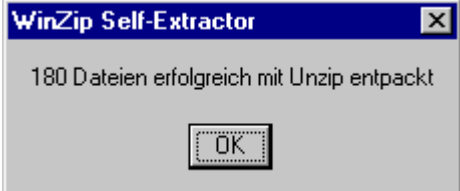
In this section of the manual, we will describe the WinCC configuration steps on the basis of the sample projects. Taking into account the multitude of potential applications WinCC has to offer, the projects described below are to be seen only as examples of what can be done with WinCC.

The WinCC projects created in this section of the manual can also be copied directly from the online document to your hard drive. By default, they will be stored to the *C:\Configuration_Manual* folder. The steps necessary to start up the WinCC projects are listed in the following table.

1.1 Downloading the Samples

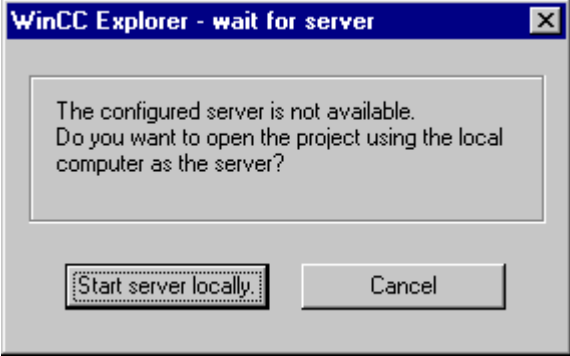

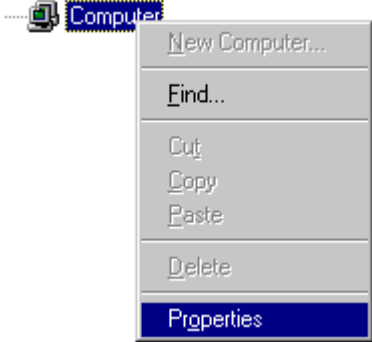
Downloading the Samples

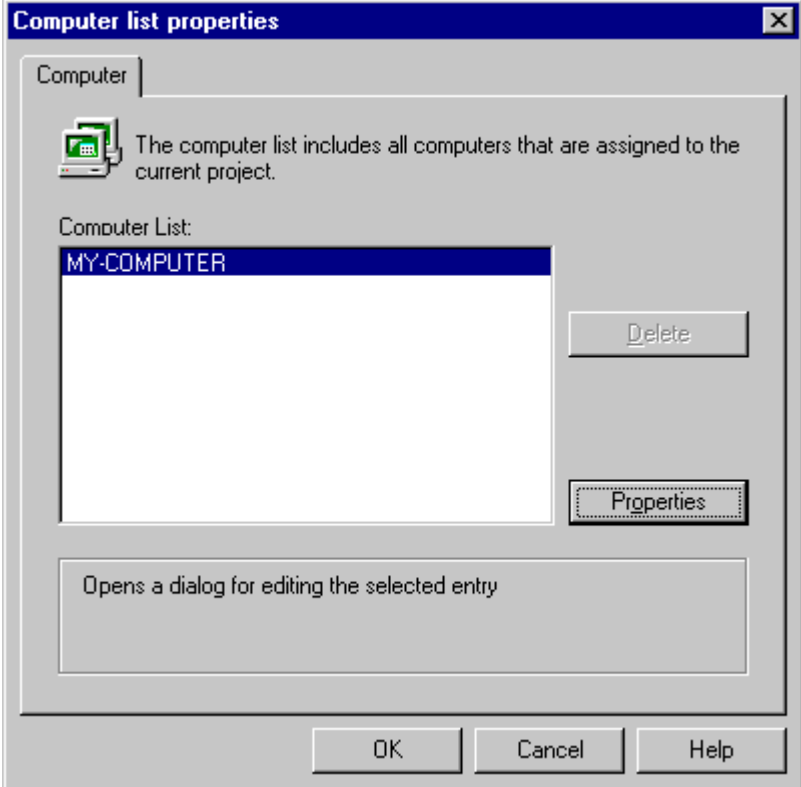
Step	Procedure: Downloading the Samples
1	Downloading the desired project. This is done from the online document by clicking on the following icon:  Project Name
2	The dialog box <i>Download File</i> will be displayed. In this dialog, select the entry <i>Execute the Program from this Location</i> . Confirm the dialog by clicking on <i>OK</i> . 

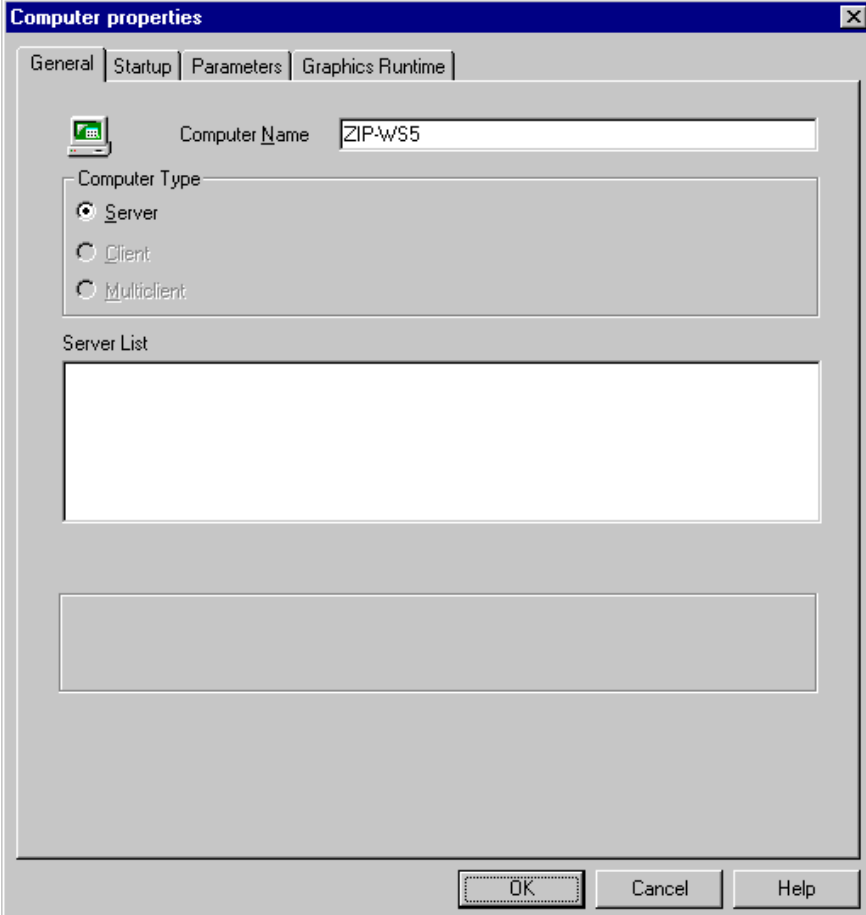
Step	Procedure: Downloading the Samples
3	<p>The dialog <i>Safety Warning</i> will be displayed. Acknowledge this dialog by clicking on <i>Yes</i>.</p> 
4	<p>The <i>WinZip Self-Extractor</i> dialog will be opened. You can specify a folder to which the project will be unzipped. By default, projects will be unzipped to the folder <i>C:\Configuration_Manual</i>. Start the unzipping process by clicking on the <i>Unzip</i> button.</p> 
5	<p>After the unzipping process has completed, a dialog box will confirm the successful unzipping of the files. Acknowledge this dialog by clicking on <i>OK</i>. The <i>WinZip Self-Extractor</i> dialog is closed via the <i>Close</i> button.</p> 

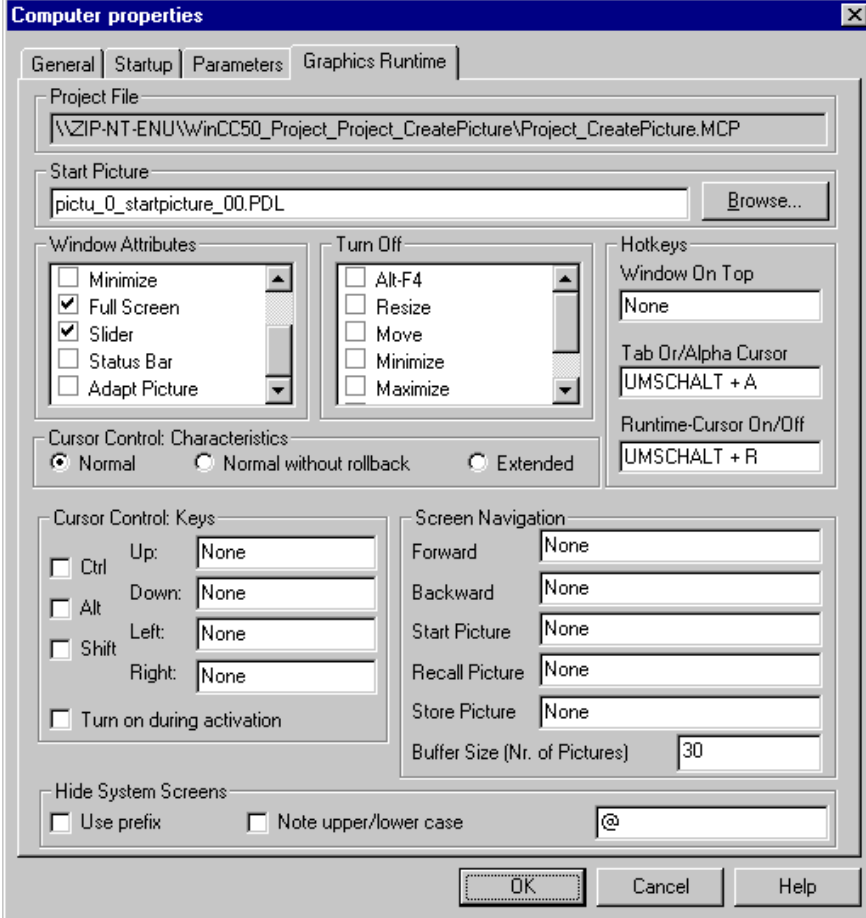
1.2 Starting Up the Samples (Single-User Projects)

Starting Up the Samples (Single-User Projects)

Step	Procedure: Starting Up the Samples (Single-User Projects)
1	<p>Open the <i>WinCC Explorer</i>. Open the sample project that has just been unzipped. A dialog box will be displayed pointing out that the server configured is not available. Via <i>Start Local Server</i>, the WinCC project is opened.</p> 
2	<p>To be able to work with the project, the name of the local server must be entered as the computer name. This is done in the <i>WinCC Explorer</i> via a  on the <i>Computer</i> entry and then selecting <i>Properties</i> from the pop-up menu.</p> 

Step	Procedure: Starting Up the Samples (Single-User Projects)
3	<p>The dialog <i>Computer List Properties</i> will be opened. The computer list will display all computers pertaining to the project. By clicking on the <i>Properties</i> button, the properties dialog of the computer is accessed.</p> 

Step	Procedure: Starting Up the Samples (Single-User Projects)
4	<p>The properties dialog of the computer will be opened. In the <i>General Information</i> tab, replace the computer entered by the local computer.</p>  <p>The screenshot shows a dialog box titled "Computer properties" with a close button (X) in the top right corner. It has four tabs: "General", "Startup", "Parameters", and "Graphics Runtime". The "General" tab is active. In the "General" tab, there is a small computer icon, a "Computer Name" text box containing "ZIP-WS5", and a "Computer Type" section with three radio buttons: "Server" (selected), "Client", and "Multiclient". Below this is a "Server List" section with an empty list box. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".</p>

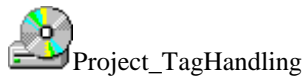
Step	Procedure: Starting Up the Samples (Single-User Projects)
5	<p>In the <i>Graphics-Runtime</i> tab, make sure that all settings are correct. Among other things, check if a start picture has been specified. If the projects are displayed using a resolution of less than 1024 x 768, the check-boxes <i>Full Screen</i> and <i>Scroll Bars</i> must be selected from the window attributes field. Exit the dialog by clicking on <i>OK</i>. Exit the computer properties dialog by clicking on <i>OK</i> as well.</p> 
6	<p>Before the project can be activated, it must be reloaded. Close the project via File → Close and then open it again.</p>

Note:

The steps just described can be applied directly to single-user projects. These steps can also be followed for the multi-user projects described in this manual, however, some additional steps must be performed which will be described in more detail in the samples concerned.

2 Tag/Variable Configuration (Project_TagHandling)

The WinCC project created in this chapter can also be copied directly from the online document to your hard drive. By default, it will be stored to the C:\Configuration_Manual folder.



In this project, you will find various tips that will make working with tags/variables easier in WinCC. Generally, WinCC deals with three different types of tags. These are Internal Tags without a process driver connection, WinCC Tags (also called External Tags) with a process driver connection and C Variables in programmed C-Actions, project functions and such. The samples pertaining to the Project_TagHandling project mainly deal with Internal Tags. The general treatment of these tags does not differ greatly from the treatment of WinCC Tags.

The samples for this topic are configured in the Project_TagHandling WinCC project. Its start page is displayed below.






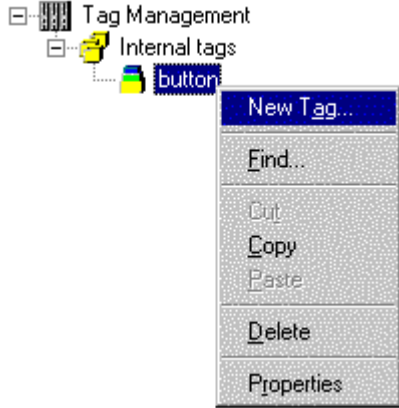
2.1 Creating, Grouping and Moving Tags

In the *WinCC Explorer*, tags can be created below the *Tag Management* entry. A distinction is made between tags without a process driver connection, so-called *Internal Tags*, and tags with a process driver connection, so-called *WinCC Tags* or *External Tags*. There is no limitation on the maximum number of configurable internal tags. The maximum number of *WinCC Tags*, however, is subject to the software license acquired.

Tag Groups and Tags

When processing large volumes of data and, consequently, a large number of tags, it is advisable to organize these tags into tag groups. Only in this way is it possible to keep a track of things in large-scale projects. The tag groups, however, do nothing toward ensuring the uniqueness of the tags. This is done solely by means of the tag names.



Step	Procedure: Tag Groups and Tags
1	<p>The creation of a tag group for <i>Internal Tags</i> is carried out in <i>Tag Management</i> via a  on the <i>Internal Tags</i> entry and then selecting <i>New Group...</i> from the pop-up menu.</p> 
2	<p>In the dialog displayed, an appropriate name must be given to the group. In the <i>WinCC Explorer</i>, a new group icon with the name just assigned will be displayed.</p> <p>In the sample project <i>Project_TagHandling</i>, the separation into groups has been made according to the chapters treated.</p>

Step	Procedure: Tag Groups and Tags
3	<p>The creation of a tag in a tag group is carried out via a  on the entry of the corresponding group and then selecting <i>New Tag...</i> from the pop-up menu.</p> 
4	<p>In the dialog displayed, assign a name to the tag in the <i>General Information</i> tab. From the list-box below, select the desired <i>Data Type</i>. It is not necessary to set an <i>Address</i> for internal tags.</p>

Note:


The current value and status of a tag in the process picture can be displayed in the WinCC Explorer via a Tooltip while runtime is active.

Moving Tags

Step	Procedure: Moving Tags
1	<p>In <i>Tag Management</i>, a tag is moved by  on it and then selecting <i>Cut</i> from the pop-up menu.</p> <p>After that, the desired target group is selected. There, the tag is inserted via a  and <i>Paste</i> from the pop-up menu.</p> <p>The same procedure can also be applied to several tags simultaneously.</p>


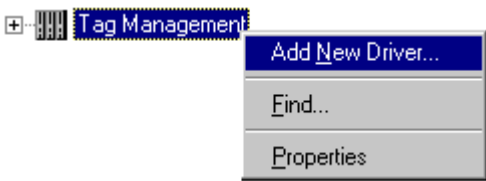
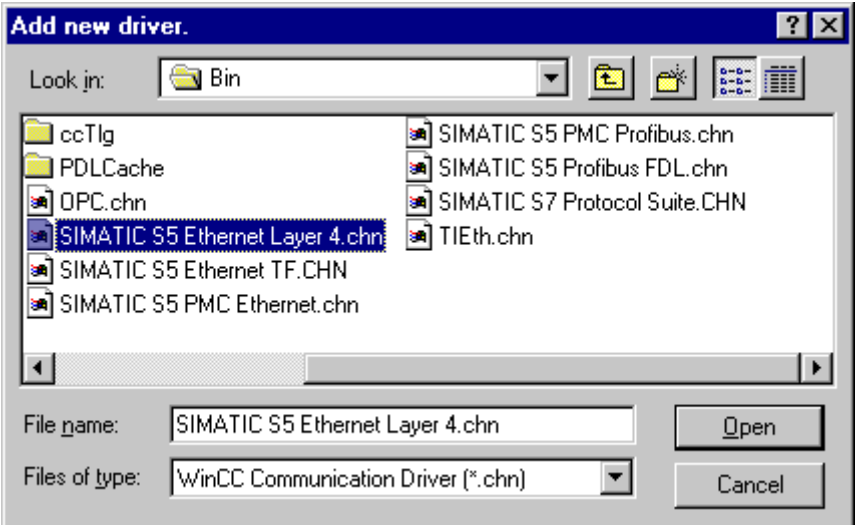
Note:



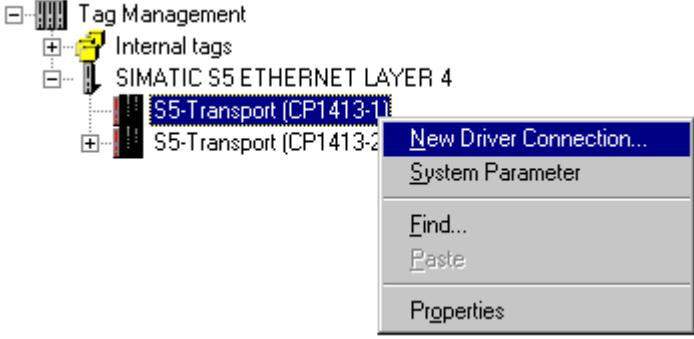

If tags are cut or deleted from the *WinCC Explorer*, runtime must not be active.

If you require a large number of tags which all have the same tag name but are numbered consecutively, you only have to create one tag of this type. This tag can be copied to the clipboard via a  and then selecting *Copy* from the pop-up menu - then tag can be inserted as often as you like. The tags will be numbered automatically in ascending order. You should take this possibility into account when defining the name convention for tags.

WinCC Tags



To create *WinCC Tags* in *Tag Management*, a connection to a PLC must be configured first. However, it is not necessary to install the hardware. It is sufficient to install the desired communication driver and to configure the desired connection.




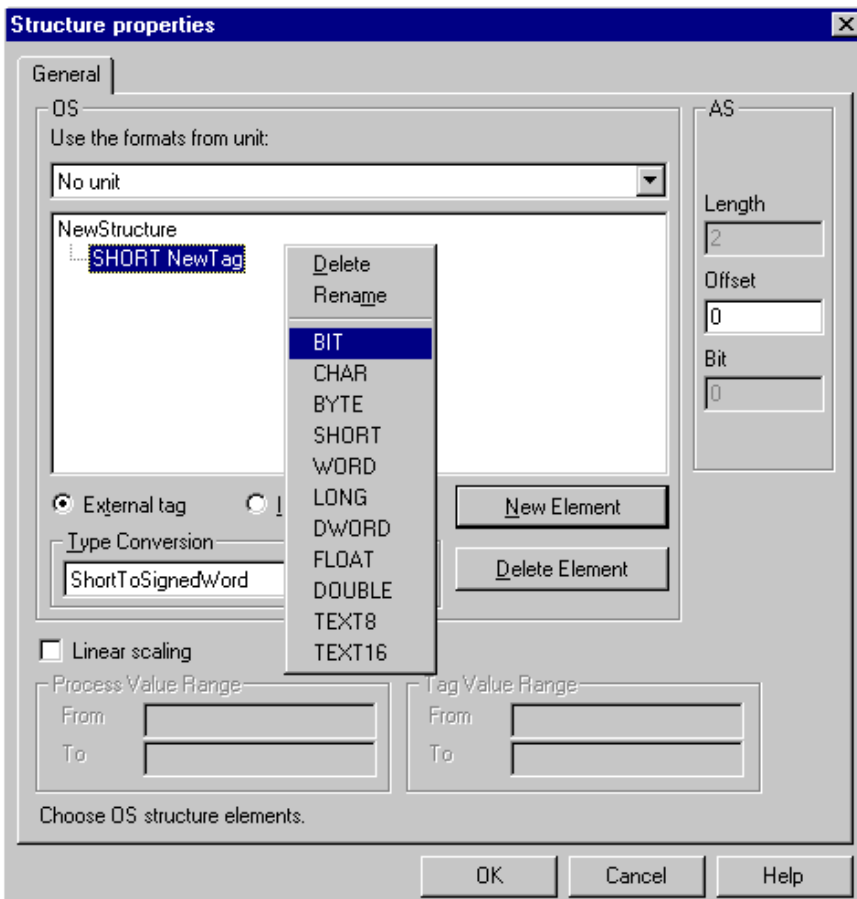
Step	Procedure: WinCC Tags
1	<p>Installation of a new communication driver. This is done via a  on the <i>Tag Management</i> entry and then selecting <i>Add New Driver...</i> from the pop-up menu.</p> 
2	<p>From the dialog displayed, select the desired driver. By clicking on the <i>Open</i> button, the driver is inserted into the WinCC project.</p> <p>The <i>WinCC Explorer</i> will now display the new driver entry in <i>Tag Management</i> in addition to the <i>Internal Tags</i>.</p> 

Step	Procedure: WinCC Tags
3	<p>Via a  on the new driver entry, one or several sub-entries, the so-called <i>Channel Units</i>, are displayed.</p> <p>Creation of a connection. This is done by  on the entry of a <i>Channel Unit</i> and then selecting <i>New Driver Connection</i> from the pop-up menu.</p> 
4	<p>In the dialog displayed, assign a name to the connection in the <i>General Information</i> tab.</p> <p>The parameters for the connection can be set by clicking on the <i>Properties</i> button.</p>
5	<p>Via a  on the newly added connection entry, tag groups and tags can be added in the manner outlined above.</p>
6	<p>When creating <i>WinCC Tags</i>, the address and adapt format settings must be defined in addition to the settings required for <i>Internal Tags</i>. The address refers to the address of the tag in the PLC.</p>

Structure Tags

Structure tags are used to group a large number of different tags and tag types that form a logical unit. These tag and tag types can then be addressed using one name. A structure tag consists of a number of individual tags, which can represent various data types.

Step	Procedure: Structure Tags
1	<p>A new structure is created via a  on the <i>Structure Type</i> entry and then selecting <i>New Structure Type</i> from the pop-up menu.</p> 

Step	Procedure: Structure Tags
2	<p>In the dialog displayed, give the structure a new name by  on the <i>NewStructure</i> entry and then selecting <i>Rename</i> from the pop-up menu.</p> 
3	<p>A new structure element can be added via the <i>New Element</i> button.</p>
4	<p>Via a  on the newly created element, its data type and name can be specified. For each structure element, you must define whether it is an <i>internal</i> or <i>external tag</i>. Clicking on the <i>OK</i> button concludes the configuration and creates the structure type.</p> 


Note:
 Once a structure type has been created, it cannot be reconfigured at a later time. The complete structure type must be defined again.

A structure tag is created in the same way as all other types of tags, however, as the data type, the created structure type must be used. The name of the individual elements of the structure tag created is composed of the structure name assigned when creating the tag and the element name assigned when creating the structure type. The two are separated in the name by a dot.

Name	Type	Parameters	Last change
 STUi_varia_str_00.aktivated	Binary Tag	Internal tag	07/10/97 02:26:14
 STUi_varia_str_00.open	Binary Tag	Internal tag	07/10/97 02:26:14
 STUi_varia_str_00.closed	Binary Tag	Internal tag	08/21/97 03:45:32
 STUi_varia_str_00.error	Binary Tag	Internal tag	07/10/97 02:26:14

2.2 Incrementing, Decrementing, Jogging

Tip. Inc. Dec

In runtime, the samples pertaining to this topic are accessed in the Project_TagHandling project by selecting the Button displayed above using the . The examples are configured in the varia_3_chapter_01.pdl and varia_3_chapter_01a.pdl pictures.

Definitions

IncrementingRefers to increasing a tag value by fixed or variable increments.

- DecrementingRefers to decreasing a tag value by fixed or variable increments.
- JoggingRefers to the execution of an action when a button is pressed, comparable to pressing a pushbutton. In the case of binary signals, this in general represents the control of a device. With analog values, a set value can be changed via jogging.


2.2.1 Jogging - Set-Point Value Change (example 01)

Task Definition

Jogging is to be performed using the mouse.

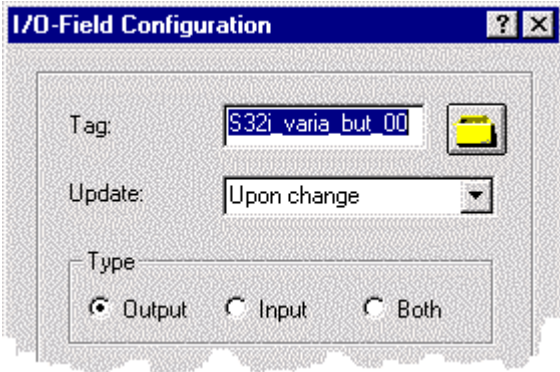

A set-point value is to be changed in fixed steps by clicking a button. This changing of the value is to be restricted to fixed limits. The changes are to be implemented locally in the picture.

Implementation Concept

For the implementation, two *Windows Object* → *Buttons* are used, with which the set-point value is changed event-driven. When the button *Button* is pressed with the , the value of an *Internal Tag* is changed by one increment. The increment is specified beforehand and cannot be altered during runtime. The set-point value change is implemented via a *C-Action*.

The change of the set-point value is displayed by a *Smart Object* → *I/O Field*. The output value of the *I/O Field* is connected to the *Internal Tag*.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Create a tag of the <i>Signed 32-Bit Value</i> type in Tag Management. In this sample, the <i>S32i_varia_but_00</i> tag is used.
2	In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field1</i> object is used. During the configuration of the <i>I/O Field</i> in the <i>configuration dialog</i> , set the <i>S32i_varia_but_00</i> tag. Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> . 
3	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button2</i> object is used.
4	To change the set-point value, create a <i>C-Action</i> at <i>Event</i> → <i>Mouse</i> → <i>Press Left</i> . This <i>C-Action</i> changes the value of the tag each time the button is clicked with the  . The limit value is specified and checked in the <i>C-Action</i> .
5	Configure the decrementation of the set-point value in the same manner. In this sample, the <i>Button1</i> object is used for this purpose.

C-Action at Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    DWORD value;

    value=GetTagDWord("S32i_varia_but_00"); //get tag value
    if (value>1300) (value=1400);          //check limit
    else value=value+100;                  //inc value
    SetTagDWord("S32i_varia_but_00",value); //set new value
}
```

- Declare the *C variable value* .
- Use the *internal function GetTagDWord* to read out the current value of the *S32i_varia_but_00* tag.
- In the *if* inquiry, check whether the value of the tag is greater than 1300. If it is, 1400 will be specified as the upper limit. If the value of the tag is less than 1300, the statement in the *else* branch is executed and the value is raised by 100.
- The *internal function SetTagDWord* then writes the changed value back into the *S32i_varia_but_00* tag.

Note for the General Application

The *C-Actions* at both *Buttons* can be used after changing the tags (internal or external), the limits and the increment.


2.2.2 Jogging - Set-Point Value Change via Global Script (example 02)

Task Definition

Jogging is to be performed using the mouse.


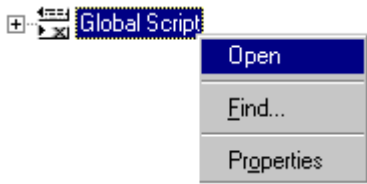
A set-point value is to be changed in fixed steps by clicking a button. This changing of the value is to be restricted to fixed limits. It is to be implemented with the aid of a *project-function*.

Implementation Concept

For the implementation, two *Windows Object* → *Buttons* are used, with which the set-point value is changed event-driven. When the button *Button* is pressed with the , the value of an *Internal Tag* is changed by one increment. The increment is specified beforehand and cannot be altered during runtime. The set-point value change is implemented via a *project function*.

The change of the set-point value is displayed by a Smart Object → *I/O Field*. The output value of the *I/O Field* is connected to the *Internal Tag*.

Creating the Project Function

Step	Procedure: Creating the Project Function
1	<p>Start the <i>Global Script</i> editor in the <i>WinCC Explorer</i> via a  on the <i>Global Script</i> entry and then selecting <i>Open</i> from the pop-up menu.</p> 
2	Create a new function via the <i>File</i> → <i>New Project Function</i> menus.
3	Assign the function name <i>IncDecValue</i> and save the function by selecting the <i>File</i> → <i>Save As</i> → <i>IncDecValue.fct</i> .
4	Program and compile the function.

Project Function IncDecValue


```

void IncDecValue(DWORD *value,DWORD low,DWORD high,DWORD step,DWORD a)
{
DWORD v;
v=*value; //get current value
switch (a){
case 0: {
if (v<step) (v=0); //low limit
else v=v-step; //decrement
} //case 0
break;
case 1: {
if (v>(high-step))
(v=high); //high limit
else v=v+step; //increment
} //case 1
break;
} //switch
*v=v; //return
}

```

- The function header with the name of the project function *IncDecValue* and the transfer parameters. The same *project function* is used for incrementing and decrementing.
- The declaration of the variable.
- When the function is called, not the variable to be processed that is transferred as the transfer parameter, but only its address. The content of this address are read into the *C variable v*.
- Using the *switch* statement, the information of the direction variable *a* is evaluated.
- In the relevant *case* branch, check the limit values and specify the maximum or minimum value if the limit is exceeded.
- If the limit is not violated, change the current value.
- Transfer the current set-point value to the address of the variable to be processed.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Create a tag of the <i>Signed 32-Bit Value</i> type in Tag Management. In this sample, the <i>S32i_varia_but_04</i> tag is used.
2	In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field2</i> object is used. During the configuration of the <i>I/O Field</i> in the <i>configuration dialog</i> , the <i>S32i_varia_but_04</i> variable is set. Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button7</i> object is used.
4	To change the set-point value, create a <i>C-Action</i> at <i>Event</i> → <i>Mouse</i> → <i>Press Left</i> . This <i>C-Action</i> calls the <i>project function IncDecValue</i> and transfers the required parameters to it. This changes the value of the tag each time the <i>button</i> is clicked with the  . The limit values are specified as the transfer parameters when the <i>project function</i> is called. The check is performed in the <i>project function</i> .

Step	Procedure: Implementation in the Graphics Designer
5	Configure the decrementation of the set-point value in the same manner. In this sample, the <i>Button6</i> object is used.

C-Action at Button7

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    DWORD value;

    value=GetTagDWord("S32i_varia_but_04");

    //IncDecValue(DWORD *value, DWORD low, DWORD high, DWORD step, DWORD a )
    IncDecValue(&value, 0, 1400, 100, 1);
    SetTagDWord("S32i_varia_but_14", value);
}
```


- Use the *internal function GetTagDWord* to read the current value of the *internal tag*.
- Call the *project function IncDecValue* and transfer the parameters (pointer to variable, lower and upper limit, increment, direction).
- Use the *internal function SetTagDWord* to transfer the changed value to the *internal tag*.

Note for the General Application

The *project function* can be used immediately without any further changes being required. In the *C-Action* used for calling the *project function*, adapt the transfer parameters to suit your own needs.

2.2.3 Jogging - Button (example 05)




The solutions pertaining to this topic are accessed in the *Project_TagHandling* project by selecting the *Buttons* displayed above using the . They are configured in the *pictu_3_chapter_01a.pdl* picture.

Task Definition

Jogging is to be performed using the mouse.
A unit (motor, valve) is to be activated by clicking a button. When the button is released, activation will be canceled.


Implementation Concept

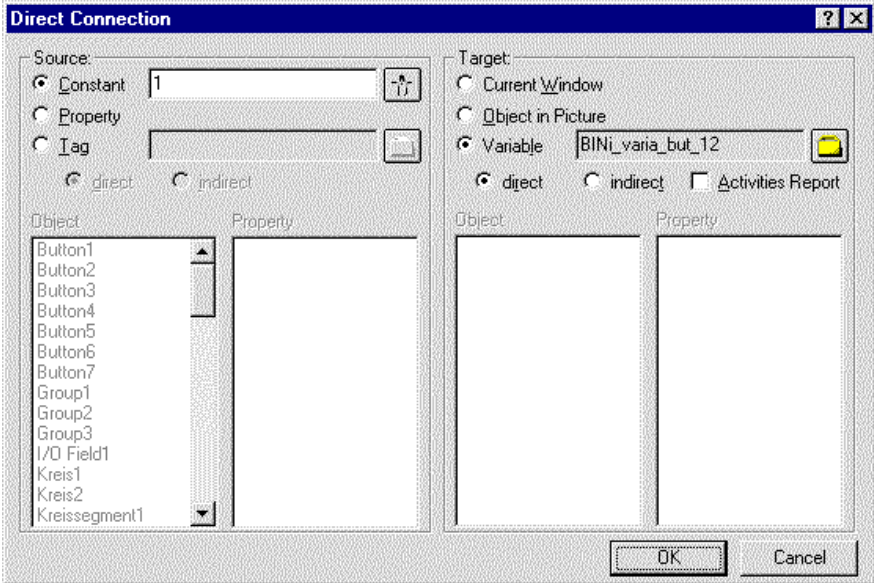
We implement the event-driven button via a *Windows Object*  *Button*.
We visualize this implementation via a *Direct Connection* and a *C-Action*.

Note:

Implementing a button via a *Direct Connection* offers the best level of performance during runtime.

Implementation in the Graphics Designer - Direct Connection

Step	Procedure: Direct Connection
1	Create a tag of the <i>Binary Tag</i> type in Tag Management. In this sample, the <i>BINi_varia_but_12</i> tag is used.
2	In a picture, we configure the <i>Windows Object</i>  <i>Button</i> . In this sample, the <i>Button2</i> object is used.

Step	Procedure: Direct Connection
3	<p>For the <i>Button2</i>, configure a <i>Direct Connection</i> at Event → <i>Mouse</i> → <i>Press Left</i>. Connect the <i>Source Constant</i> → <i>1</i> to the <i>Target Tag</i> → <i>BINi_varia_but_12</i>. Apply the settings by clicking on the <i>OK</i> button. Configure another <i>Direct Connection</i> at Event → <i>Mouse</i> → <i>Release Left</i>, but this time for the <i>Source Constant</i> → <i>0</i>.</p> 
4	The animation is controlled via the <i>BINi_varia_but_12</i> tag.

Optionally, the implementation of the same task is explained below using a *C-Action*. The implementation outlined above using a *Direct Connection* is the better and faster approach.

Implementation in the Graphics Designer - C-Action

Step	Procedure: C-Action
1	Create a tag of the <i>Binary Tag</i> type in Tag Management. In this sample, the <i>BINi_varia_but_12</i> tag is used.
2	In a picture, we configure the <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button1</i> object is used.
3	At Event → <i>Mouse</i> → <i>Press Left</i> , create a <i>C-Action</i> which sets the value of the <i>BINi_varia_but_12</i> tag to <i>1</i> . At Event → <i>Mouse</i> → <i>Release left</i> , create another <i>C-Action</i> which sets the value of the <i>BINi_varia_but_12</i> tag to <i>0</i> .

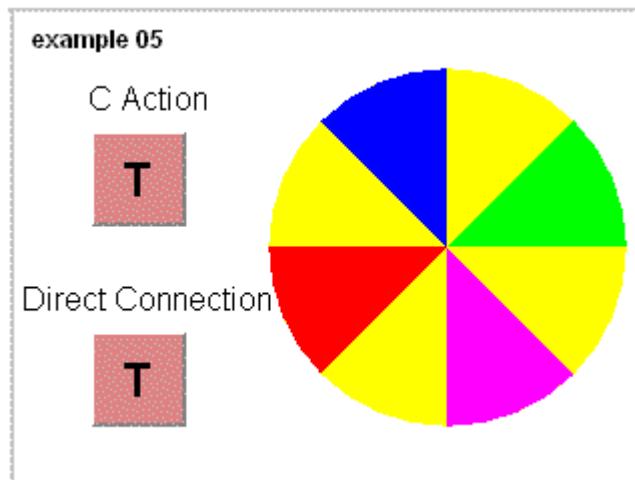
C-Action at Button1

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetTagWord("BINi_varia_but_12",1); //on
}
```

- Use the *internal function SetTagDWord* to set the tag to 1.

Animation of the Sample

In this sample, we use the button to animate the following color wheel.



- The color wheel consists of several *Standard Objects* → *Pie Segments*.
- All objects are made dynamic using a *Dynamic Dialog at Properties* → *Geometry* → *Start Angle* and at *Properties* → *Geometry* → *End Angle*. To change the value, we need an action which changes the value of the rotation angle at fixed time slots. We implement this value change via a *C-Action* for the *Pie Segment4 at Property* → *Colors* → *Line Color*. As the trigger for the action, we set 250 ms. In this case, we do **not** make the *Line Color* dynamic. The reason for the *C-Action* at this property is that we require a trigger for the value change. We could also use a different property of the object instead of the
- The current rotation angle is changed in the *Internal tag S32i_vara_but_11*.

C-Action for the Animation

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN
{
    Static DWORD i = 0;

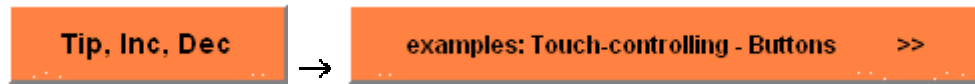
    //if button pressed
    if (GetTagBit("BINi_varia_but_12")) {
        i=i+10; //increment of rotation
        if (i==360) (i=0); //high limit
        SetTagDWord("S32i_varia_but_11",i);
    }//if
    return(0x0); //black
}
```


- Declare the *C* variable *i* as a *static DWORD*, since its value must remain constant while the picture is open.
- If the *Button* is pressed (button on), the wheel is rotated in increments of 10 degrees, i.e. the value of the tag is incremented by 10.
- The tag *i* is initialized once the wheel has been rotated completely (360°).
- Transfer the new value for the rotation angle of the *internal tag*.
- Return the configured value of the background color with *return*. It is not supposed to be changed.

Note for the General Application

The *Button* with the *Direct Connection* can be used after changing the tag.

2.2.4 Jogging - Changeover Switch (example 06)



The example pertaining to this topic is accessed in the *Project_TagHandling* project by selecting the *Buttons* displayed above using the . It is configured in the *pictu_3_chapter_01a.pdl* picture.


Task Definition

Jogging is to be performed using the mouse.

The function of a changeover switch is to be implemented by means of a button.

Pressing the button will switch the unit (motor, valve) on and the unit is to remain on once the button is released. Pressing the button again will switch the unit off.









Implementation Concept

We implement the event-driven changeover switch via the Windows Object  Button.

Note:

Implementing a changeover switch via a *Direct Connection* offers the best performance during runtime, but requires two buttons.

Implementation in the Graphics Designer - Direct Connection

Step	Procedure: Direct Connection
1	Create a tag of the <i>Binary Tag</i> type in Tag Management. In this sample, the <i>BINi_varia_but_16</i> tag is used.
2	In a picture, configure two <i>Windows Objects</i>  <i>Buttons</i> . In this sample, the <i>Button4</i> object is used for switching on and the <i>Button5</i> object for switching off.
3	For the <i>Button4</i> , configure a <i>Direct Connection</i> at <i>Event</i>  <i>Mouse</i>  <i>Press Left</i> . Connect the <i>Source Constant</i>  <i>1</i> to the <i>Target Tag</i>  <i>BINi_varia_but_16</i> . Apply the settings by clicking on the <i>OK</i> button. For the <i>Button5</i> , configure a <i>Direct Connection</i> as outlined above, but with the <i>Source Constant</i>  <i>0</i> .
4	The <i>direct connection at Event</i>  <i>Mouse</i>  <i>Mouse Action</i> only synchronizes the labeling on the <i>Button3</i> and is not required for the functionality of the changeover switch.

Implementation in the Graphics Designer - C-Action

Step	Procedure: C-Action
1	Create a tag of the <i>Binary Tag</i> type in Tag Management. In this sample, the <i>BINi_varia_but_16</i> tag is used.
2	In a picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button3</i> object is used.
3	At Event → <i>Mouse</i> → <i>Press Left</i> , create a <i>C-Action</i> that negates the status of the <i>BINi_varia_but_16</i> tag.

C-Action for the Changeover Switch

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    BOOL state;

    //flip tag
    state = !GetTagBit("BINi_varia_but_16");
    SetTagBit("BINi_varia_but_16", (SHORT)state);
}
```

- The declaration of the *state* variable.
- Via the *internal function GetTagBit*, the value of the *internal tag* is read, inverted and then returned via the *SetTagBit* function.


Note for the General Application

The button with the *C-Action* can be used after changing the variable. The inversion of the *internal tag* can also be performed without the *C variable* as shown below:

```
SetTagDWord("BINi_varia_but_16",
(SHORT)!GetTagBit("BINi_varia_but_16"));
```

2.2.5 Incrementing and Decrementing (example 01)


Tip, Inc, Dec



The example pertaining to this topic is accessed in the *Project_TagHandling* project by selecting the *Button* displayed above using the . It is configured in the *pictu_3_chapter_01.pdl* picture.


Task Definition

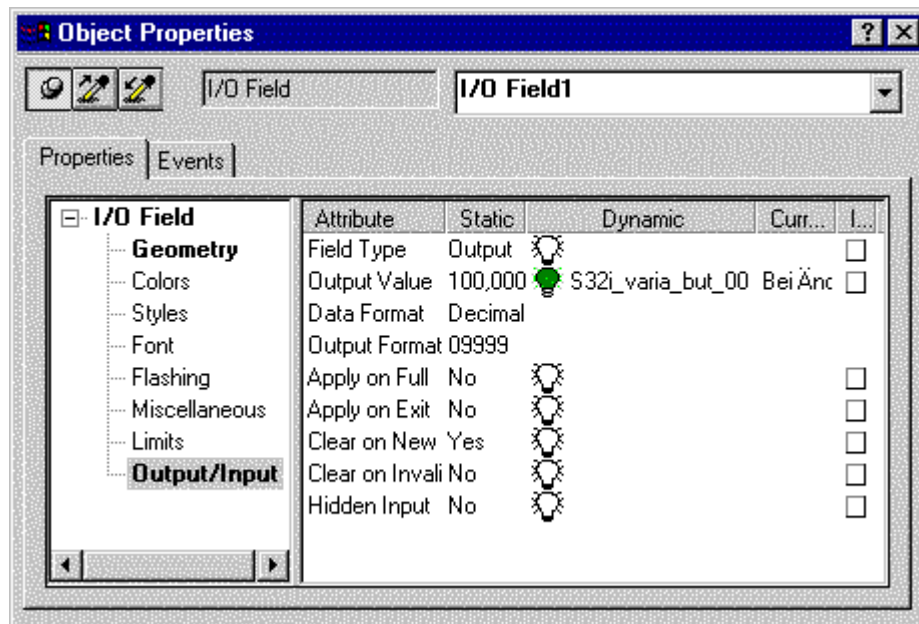
The value of a tag is to be changed. This changing of the value is to be restricted to fixed limits. Changing of the value is to be performed using the mouse.
A tag value is to be changed by pressing a button. The value is to be changed only when the button is pressed. The value set must be retained when the button is released.

Implementation Concept

For the implementation of the event-driven button, a Windows Object  Button is used.

When the button is pressed with the , the value of an *internal tag* is increased by the increment set, when the button is pressed with the , the value is decreased by the increment set. The value keeps changing as long as the button is pressed. The increment is specified beforehand and cannot be altered during runtime.



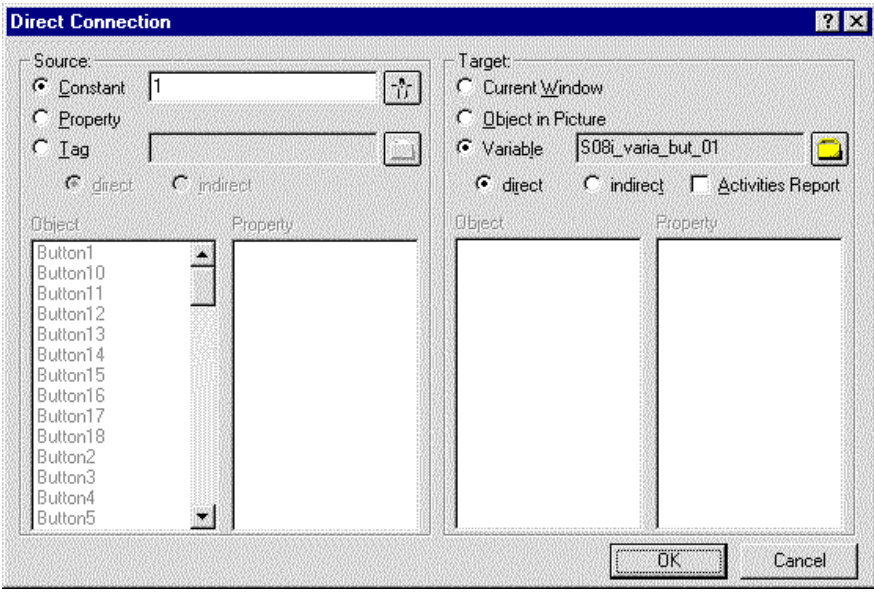
To display the value change, a *Smart Object*  *I/O Field* is used. The output value of the *I/O Field* is connected to the *Internal Tag*.

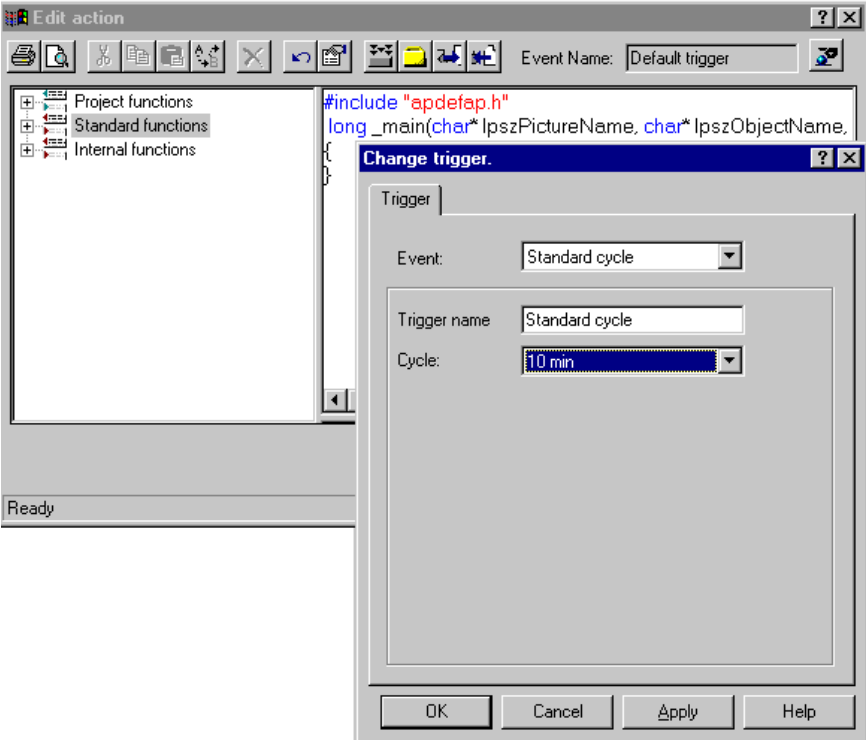


Changing the Value

For the value change, an action is required that changes the value of an *internal tag* in fixed time slots. The value change is implemented with a *C-Action* directly at the *Property* → *Geometry* → *Position X* of the *I/O Field*. As the trigger for the action, we set *250 ms*. We are **not** making the position of the *I/O Field* dynamic. The reason for the *C-Action* at this property is that we want to implement the value change directly at the object. In this sample project, we have also solved this problem by using a *Global Action*.

Implementation in the WinCC project

Step	Procedure: Incrementing, Decrementing
1	Creating the tag in the Tag Management. In this sample, the <i>S32i_varia_but_00</i> and <i>S08i_varia_but_01</i> tags are used.
2	In a picture, configure a <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field1</i> object is used. During the creation of the <i>I/O Field</i> , set the <i>S32i_varia_but_00</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button3</i> object is used.
4	For the set-point value change via a mouse click, several <i>direct connections</i> are created at this <i>Button</i> . These <i>direct connections</i> change the value of the <i>S08i_varia_but_01</i> tag each time the <i>Button</i> is pressed via a  or  . At Event → Mouse → Press Left, set the incrementation ON (set tag to 1). At Event → Mouse → Release left, set the incrementation OFF (set the tag to 0). At Event → Mouse → Press Right, set the decrementation ON (set tag to 2) and at Event → Mouse → Release Right, set the decrementation OFF (set tag to 0).
	 <p>The screenshot shows the 'Direct Connection' dialog box. On the left, under 'Source', the 'Constant' radio button is selected with a value of '1'. Below it, 'Object' is set to 'Button3'. On the right, under 'Target', the 'Variable' radio button is selected with the variable name 'S08i_varia_but_01'. Below it, 'Object' is also set to 'Button3'. The 'direct' radio button is selected for both source and target. There are 'OK' and 'Cancel' buttons at the bottom right.</p>
5	The value change of the <i>S32i_varia_but_00</i> tag is carried out in a <i>C-Action</i> at the

Step	Procedure: Incrementing, Decrementing
	<i>Property</i> → <i>Geometry</i> → <i>Position X</i> of the object <i>I/O Field1</i> .
6	<p>The trigger for calling the <i>C-Action</i> is changed to 250 ms.</p> 

C-Action at the I/O Field for the Value Change

```

#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    DWORD value;
    SHORT count;

    count = GetTagWord("S08i_varia_but_01"); //inc or dec
    if ((count==1) || (count==2)){
        //current value
        value = GetTagDWord("S32i_varia_but_00");

        if (count==1){ //inc
            value++;
            if (value>1400) (value=1400); //high limit
            SetTagDWord("S32i_varia_but_00",value);
        }//inc
        if (count==2){ //dec
            value--;
            if (value<0) (value=0); //low limit
            SetTagDWord("S32i_varia_but_00",value);
        }//dec
    }//if count
    return(81); //x-pos
}

```

- The declaration of the *C variables value* and *count*.
- Evaluation of whether the *Button* is pressed. If the *Button* is not pressed, the *C-Action* is ended (to avoid unnecessary system loads).
- If the *Button* is pressed, the script inquires whether the value is to be incremented or decremented. The value of the tag is changed depending on the result of this evaluation.
- After the value has been changed, the limit value check is performed.
- Return the value configured for position X with *return*. It is not supposed to be changed.

Note for the General Application

The button with the *direct connections* can be used after changing the tags and in conjunction with the *C-Action* at the *I/O Field*. In the *C-Action*, the limit values and variables must be adapted.

2.2.6 Incrementing and Decrementing via Global Script (example 02)



Task Definition

The value of a tag is to be changed. This changing of the value is to be restricted to fixed limits. Changing of the value is to be performed using the mouse.

A tag value is to be changed by pressing a button. The value is to be changed only when the button is pressed. The value set must be retained when the button is released.

Implementation Concept

For the implementation of the event-driven button, a Windows Object → Button is used. The implementation is carried out via a *Global Action*.

When the button is pressed with the , the value of an *internal tag* is increased by the increment set, when the button is pressed with the , the value is decreased by the increment set. The value keeps changing as long as the button is pressed. The increment is specified beforehand and cannot be altered during runtime.

To display the value change, a *Smart Object* → *I/O Field* is used. The output value of the *I/O Field* is connected to the *Internal Tag*.

Changing the Value

For the value change, an action is required that changes the value of an *internal tag* in fixed time slots. We implement the value change via a *Global Action*.

The action is activated when the WinCC runtime is started and is then processed with the set cycle. The action is programmed in such a way that the actual program component is only processed when the button is pressed.

One unusual feature of this action is that it uses external C variables. External C variables are recognized throughout the entire WinCC runtime, but they must be declared outside of the function header. Since in WinCC this is only possible in a project function, a separate project function is created for the declaration of these tags. This project function must be executed once when the project is started and is then no longer required.

Creating the Project Function

Step	Procedure: Creating the Project Function
1	In the <i>WinCC Explorer</i> , start the <i>Global Script</i> editor.
2	Create a new function via the <i>File</i> → <i>New Project Function</i> menus.
3	Assign the <i>InitAction</i> function name and save the function by selecting <i>File</i> → <i>Save As</i> → <i>InitAction.fct</i> .
4	Program and compile the function.


Project Function InitAction

```
//declaration for counter.pas
extern char tagname[30] = " ";
extern SHORT count = 0;
extern DWORD low = 0;
extern DWORD high = 0;
extern DWORD step = 0;

void InitAction()
{
//function is used to generate external tags
}
```

- The declaration of the external *C variables*.
- This function must be executed once when the project is started and is then no longer required. Its execution is recommended in the start picture at *Event* → *Miscellaneous* → *Open Picture*.

Creation of the Global Action

Step	Procedure: Creation of the Global Action
1	In the <i>WinCC Explorer</i> , start the <i>Global Script</i> editor.
2	Create a new action via the <i>File</i> → <i>New Action</i> menus.
3	Save the file by selecting <i>File</i> → <i>Save As</i> → <i>counter.pas</i> .
4	Program and compile the action.
5	Set the trigger. This is done via the <i>Button</i>  on the toolbar. In the <i>Description</i> dialog window, select the <i>Trigger</i> tab. Add the <i>Timer</i> → <i>Standard Cycle</i> → <i>250 ms</i> .

Global Action counter.pas

```
#include "apdefap.h"



int gscAction( void )
{
extern char tagname[30];
extern SHORT count;
extern DWORD low;
extern DWORD high;
extern DWORD step;

DWORD value;

if ((count==1)|| (count==2)) {
//get current value
value = GetTagDWord(tagname);
if (count==1){ //inc
value = value+step;
if (value>high) (value=high); //high limit
} //if
if (count==2){ //dec
value = value-step;
if (value<low) (value=low); //low limit
} //if
SetTagDWord(tagname,value);
} //if
return(0);
}
```

- The declaration of the external *C variable*.
- Evaluation of whether the *Button* is pressed. If the *Button* is not pressed, the *C-Action* is ended (to avoid unnecessary system loads).
- If the *Button* is pressed, the script inquires whether the value is to be incremented or decremented. Depending on the evaluation result, the value of the *C variable value* is changed.
- After the value has been changed, the limit value check is performed.
- Use the *internal function SetTagDWord* to assign the new value to the variable to be processed.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Creating the tag in the Tag Management. In this sample, the <i>S32i_varia_but_04 tag is used</i> .
2	In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field2</i> object is used. During the creation of the <i>I/O Field</i> , set the <i>S32i_varia_but_04</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button8</i> object is used.
4	For the set-point value change via a mouse click, several <i>C-Actions</i> are created at this <i>Button</i> . At <i>Event</i> → <i>Mouse</i> → <i>Press Left</i> , set the incrementation ON and at <i>Event</i> → <i>Mouse</i> → <i>Release Left</i> , set the incrementation OFF. At <i>Event</i> → <i>Mouse</i> → <i>Press Right</i> , set the decrementation ON and at <i>Event</i> → <i>Mouse</i> → <i>Release Right</i> , set the decrementation OFF. These <i>C-Actions</i> supply the <i>Global Action counter.pas</i> with the appropriate parameters. This happens every time the button is clicked via a  or  .
5	The value change of the <i>S32i_varia_but_04</i> tag is carried out in the <i>Global Action counter.pas</i> .

C-Action at Button8 for Incrementation ON

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
//inc on
extern char tagname[30];
extern SHORT count;
extern DWORD low;
extern DWORD high;
extern DWORD step;

strcpy(tagname, "S32i_varia_but_04");
count = 1;
low = 0;
high =1400;
step = 1;
}
```

C-Action at Button8 for Incrementation OFF

```
#include "apdefap.h"
void OnLButtonUp(char* lpszPictureName, char* lpszObjectName, char* lpszPrp
{
  //inc off
  extern SHORT count;
  count=0;
}
```

- The declaration of the external *C variables* in the *C-Action*. These variables are generated by the *InitAction* project function.
- The variables are supplied with the corresponding values. This is comparable to the transfer of parameters to a *project function*. The content of the *count* variable is responsible for processing the program in the *Global Action*.
- When switching off the incrementation process, there is no need to set all the tags.

Note for the General Application

The following adaptations must be made before the general application:

- In the *C-Actions*, change the tag and adapt the limit values and the increment.
- If this button is transferred to another project, the *project function InitAction* as well as the *Global Action counter.pas* must both be transferred together with the button.

2.2.7 The remaining Samples of this Topic

example 03

The functionality of this sample is similar to that of sample *example 01*. The basic difference is that the increment can be changed during runtime. Another difference is the dynamic changing of the increment when the increment is being set. If the increment is > 20, the value is changed in steps of 10; if the increment is < 20, the value is changed in steps of 1.


example 04

The functionality of this sample is the result of the combination of the samples *example 01* and *example 02*. The value is changed with the help of the *Global Action counter.pas*.


example 07

The functionality of this sample is similar to that of sample *example 05*. The difference here is in the mode of animation.

2.3 Changing Tag Values via Windows Objects

An orange rectangular button with the text "Win Objects" in white, bold font. There are small white dots on the left and right sides of the button, suggesting it is a scrollable list item.

Win Objects

In runtime, the solutions pertaining to this topic are accessed in the *Project_TagHandling* project by selecting the *Button* displayed above using the . The samples are configured in the *varia_3_chapter_02.pdl* picture.

2.3.1 Input via a Slider with Direct Connection (example 01)

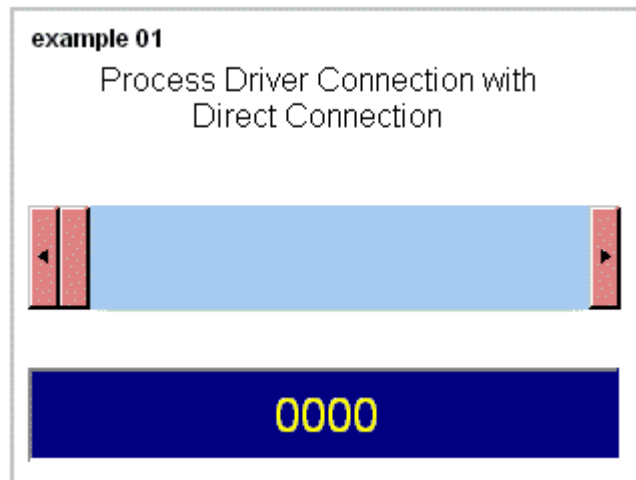
Task Definition

Changing a set-point value is to be performed via a slider.
This changing of the value is to be restricted to fixed limits.

Implementation Concept

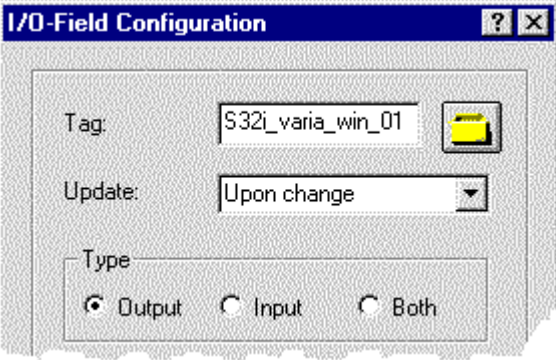
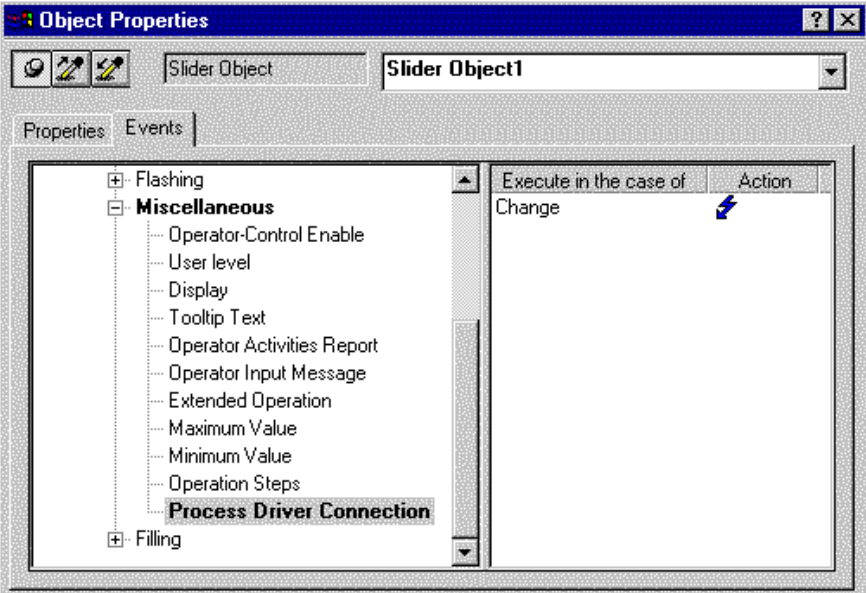
For the implementation of the set-point value change, we will use a *Windows Object* → *Slider Object*. Via a *direct connection*, the value of an *internal tag* is changed when the position of the slider is changed.

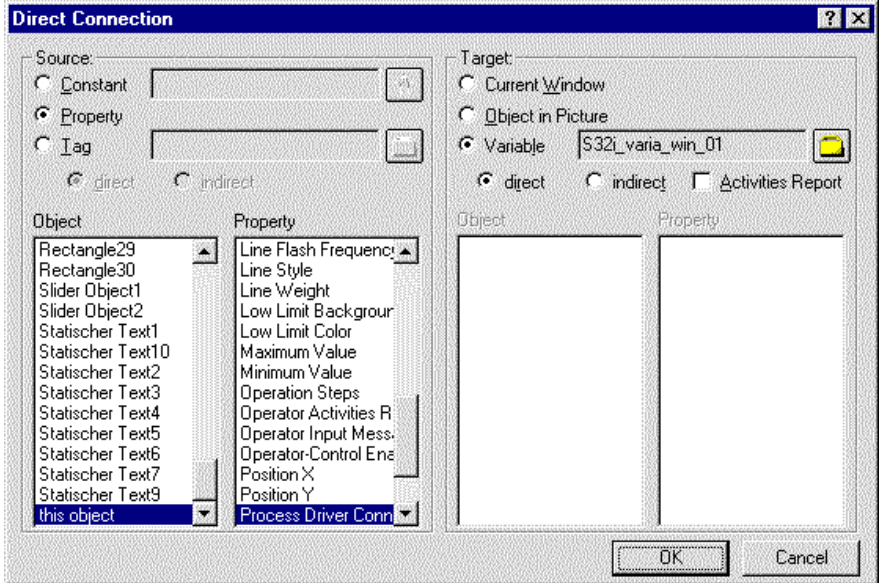
The set-point value change is displayed in a Smart Object → I/O Field.



Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Create a tag in Tag Management. In this sample, the <i>S32i_varia_win_01</i> tag is used.

Step	Procedure: Implementation in the Graphics Designer
2	<p>In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i>. In this sample, the <i>I/O Field1</i> object is used. During the creation of the <i>I/O Field</i>, set the <i>S32i_varia_win_01</i> tag in the <i>configuration dialog</i>. Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i>.</p> 
3	<p>In the same picture, configure a <i>Windows Object</i> → <i>Slider Object</i>. In this sample, the <i>Slider Object1</i> is used. At <i>Event</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i>, create a <i>direct connection</i>.</p> 

Step	Procedure: Implementation in the Graphics Designer
4	<p>In the <i>Direct Connection</i> dialog, connect the <i>source this object</i> → <i>Process Driver Connection</i> with the <i>target Variable</i> → <i>S32_varia_win_01</i>. Apply the settings by clicking on the <i>OK</i> button.</p>
	

Note for the General Application

The following adaptations must be made before the general application:

- Change the *direct connection tag*.
- The value range of the *Slider Object* can be changed via *Properties* → *Miscellaneous* → *Maximum Value* and *Minimum Value*. This can also be done in the slider's *configuration dialog*.

2.3.2 Input via a Slider and Tag Connection (example 03)

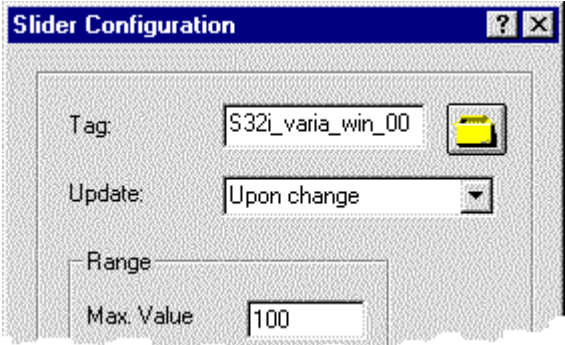
Task Definition

Changing a set-point value is to be performed via a slider.
This changing of the value is to be restricted to fixed limits.

Implementation Concept

For the implementation of the set-point value change, we will use a *Windows Object* → *Slider Object*. Via a *tag connection*, the value of an *internal tag* is changed when the position of the slider is changed. The tag is only written to when the slider is released. The set-point value change is displayed in a *Smart Object* → *I/O Field*.

Implementation in the WinCC project

Step	Procedure: Set-Point Value Change via a Slider - Tag Connection
1	Create a tag in Tag Management. In this sample, the <i>S32i_varia_win_00</i> tag is used.
2	In a picture, configure a <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field3</i> object is used. During the creation of the <i>I/O Field</i> , set the <i>S32i_varia_win_00</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Slider Object</i> . In this sample, the <i>Slider Object2</i> is used. During the creation of the <i>Slider Object</i> , set the <i>S32i_varia_win_00</i> tag in the <i>configuration dialog</i> . Change the <i>Update</i> default value from 2 s to <i>Upon Change</i> . 

Note for the General Application

The following adaptations must be made before the general application:


- Change the tag in the *tag connection*.
- The value range of the *Slider Object* can be changed via *Properties* → *Miscellaneous* → *Maximum Value* and *Minimum Value*. This can also be done in the slider's *configuration dialog*.

2.3.3 Input via an Option Group (Radio-Button) (example 02)

Task Definition

Changing of a set-point value is to be implemented by selecting specified, fixed values from a list.

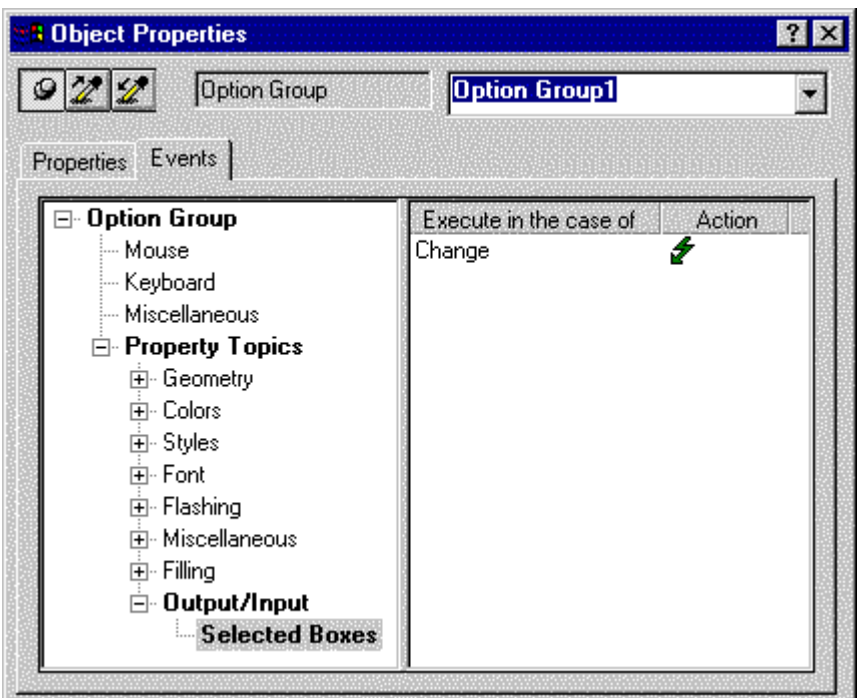
Implementation Concept

For the implementation of the set-point value change, we will use the *Windows Object* → *Option Group*. When one of the specified set-point values is selecting via a , the value in an *internal tag* is changed. The list of set values is specified and cannot be altered during runtime.

The change of the set-point value is displayed by a *Smart Object* → *I/O Field*. The output value of the *I/O Field* is connected to the *Internal Tag*. The set-point value change is implemented via a *C-Action*.

Implementation in the WinCC project

Step	Procedure: Set-Point Value Change via an Option Group
1	Create a tag in Tag Management. In this sample, the <i>S32i_varia_win_02</i> tag is used.
2	In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field2</i> object is used. During the creation of this <i>I/O Field</i> , set the <i>S32i_varia_win_02</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Option Group</i> . In the sample, this is the <i>Option Group1</i> . At the <i>Property</i> → <i>Geometry</i> → <i>Number of Boxes</i> , change the default value 3 to 4.
4	Select the Index 1 via <i>Properties</i> → <i>Font</i> → <i>Index</i> → 1. Enter the appropriate text for the selected index at <i>Properties</i> → <i>Font</i> → <i>Text</i> → 0. In the same way, configure the values for the remaining index inputs.

Step	Procedure: Set-Point Value Change via an Option Group
5	<p>At Events → Property Topics → Output/Input → Selected Boxes, create a C-Action which sets the S32i_varia_win_02 tag to a certain value depending on the selected radio-button.</p> 

C-Action at the Option Group

```

Void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* lpszE
{
//set tag according to selected box
switch(value){
case 1 : SetTagDWord("S32i_varia_win_02",0);
break;
case 2 : SetTagDWord("S32i_varia_win_02",50);
break;
case 4 : SetTagDWord("S32i_varia_win_02",100);
break;
case 8 : SetTagDWord("S32i_varia_win_02",150);
break;
} //switch
}

```

- Assign values to the *S32i_varia_win_02* tag according to the input status. The input status is stored in the predefined *value* tag.

Note for the General Application

The following adaptations must be made before the general application of the *Option Group*:

- Adapt the tag in the *C-Action at Events* → *Property Topics* → *Output/Input* → *Selected Boxes*.

2.3.4 Input via a Check-Box (example 04)

Task Definition

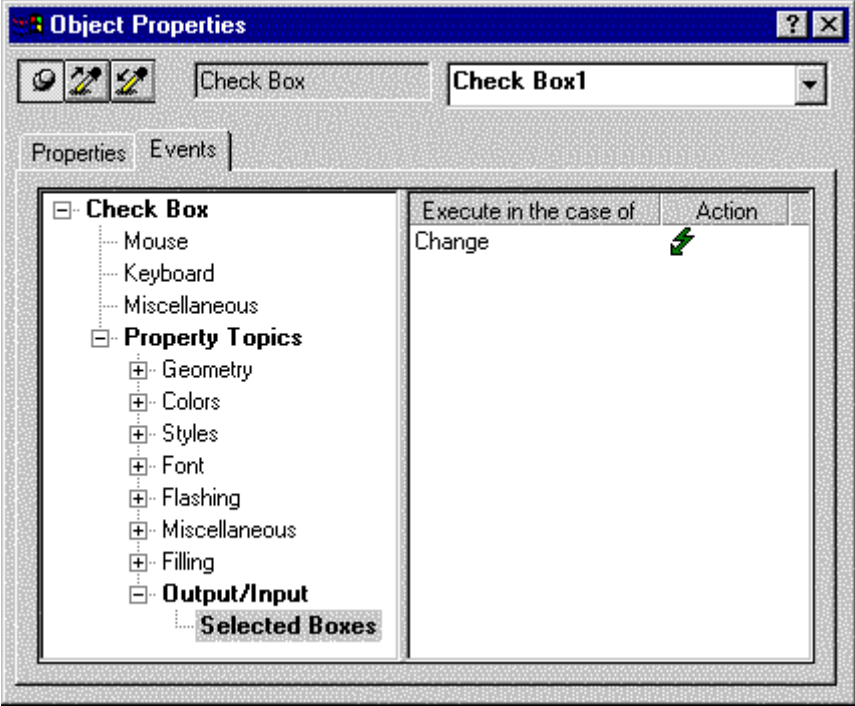
Via a check-box, various objects are to be displayed or hidden.

Implementation Concept

For the implementation, a *Windows Object* → *Check-Box* is used, which sets the individual Bits of a tag. A number of *Standard Objects* → *Polygons* are displayed or hidden, depending on these bits. To display the binary output value of the check-box, a *Smart Object* → *I/O Field* is used.

Implementation in the WinCC project

Step	Procedure: Input via a Check-Box
1	Create a tag of the <i>Signed 32-Bit Value</i> type in Tag Management. In this sample, the <i>S32i_varia_win_03</i> tag is used.
2	<i>Configure several Standard Objects</i> → <i>Polygons</i> . In this sample, <i>Polygon1</i> to <i>Polygon7</i> are used. These objects are to be displayed or hidden depending on the selection status of the <i>Check-Box</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Check-Box</i> . In this sample, this is the <i>Check-Box1</i> . At <i>Property</i> → <i>Geometry</i> → <i>Number of Boxes</i> , change the default value 3 to 7.
4	Select the <i>Index 1</i> via <i>Properties</i> → <i>Font</i> → <i>Index</i> → <i>1</i> . Enter the appropriate text for the selected index at <i>Properties</i> → <i>Font</i> → <i>Text</i> . This text being the name of the object that you want to control by selecting this check-box. In the same way, configure the values for the remaining index inputs.

Step	Procedure: Input via a Check-Box
5	<p>At <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Selected Boxes</i>, create a <i>C-Action</i>, which assigns the binary status <i>Check-Box1</i> to the <i>S32i_varia_win_03</i> tag and controls the display of the individual <i>Polygon</i> objects.</p>  <p>The screenshot shows the 'Object Properties' dialog box for a 'Check Box' object. The 'Events' tab is selected. The left pane shows a tree view of 'Property Topics' with 'Output/Input' expanded to 'Selected Boxes'. The right pane shows an event table with 'Change' in the 'Execute in the case of' column and a lightning bolt icon in the 'Action' column.</p>
6	<p>Configure a <i>Smart Object</i> → <i>I/O Field</i>. In this sample, the <i>I/O Field4</i> object is used. In the <i>configuration dialog</i>, set the <i>S32i_varia_win_03</i> tag. Change the <i>Update</i> default value from 2 s to <i>Upon Change</i>. At <i>Properties</i> → <i>Output/Input</i>, change the <i>Data Format</i> to <i>Binary</i> and the <i>Output Format</i> to <i>01111111</i>.</p>

C-Action at the Check-Box

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
SetTagDWord("S32i_varia_win_03", value);
//first box selected
if (value&1) SetVisible(lpszPictureName, "Polygon1", 1);
else SetVisible(lpszPictureName, "Polygon1", 0);
//second box selected
if (value&2) SetVisible(lpszPictureName, "Polygon2", 1);
else SetVisible(lpszPictureName, "Polygon2", 0);
//third box selected
if (value&4) SetVisible(lpszPictureName, "Polygon3", 1);
else SetVisible(lpszPictureName, "Polygon3", 0);
//fourth box selected
if (value&8) SetVisible(lpszPictureName, "Polygon4", 1);
else SetVisible(lpszPictureName, "Polygon4", 0);
//fifth box selected
if (value&16) SetVisible(lpszPictureName, "Polygon5", 1);
else SetVisible(lpszPictureName, "Polygon5", 0);
//sixth box selected
if (value&32) SetVisible(lpszPictureName, "Polygon6", 1);
else SetVisible(lpszPictureName, "Polygon6", 0);
//seventh box selected
if (value&64) SetVisible(lpszPictureName, "Polygon7", 1);
else SetVisible(lpszPictureName, "Polygon7", 0);
}
```

- Set the *S32i_varia_win_03* tag to the new input status of the *Check-Box*.
- Control the visibility of the objects in accordance with the input status. The input status is stored in the predefined *value* tag. To read out the respective bit, you have to perform bit masking to the relevant bit.

Note:

A similar sample is shown in the *Project_CreatePicture* project, chapter *Adding Dynamics, example4*. In that case, however, visibility is queried for each individual object via a *Dynamic Dialog*.


Note for the General Application

The following adaptations must be made before the general application of the *Check-Box*:

- Adapt the variable and the object names in the *C-Action at Event* → *Property Topics*
→ *Output/Input* → *Selected Boxes*.

2.4 Bit Processing in Words



The solutions pertaining to this topic are accessed in the Project_TagHandling project by selecting the Button displayed above using the . The samples are configured in the varia_3_chapter_03.pdl and varia_3_chapter_03a.pdl pictures.

Definition


The term **Bit Processing** refers to changing the status of Bits in a word.

2.4.1 Setting a Bit directly via a Check-Box and Direct Connection (example 06)

Task Definition

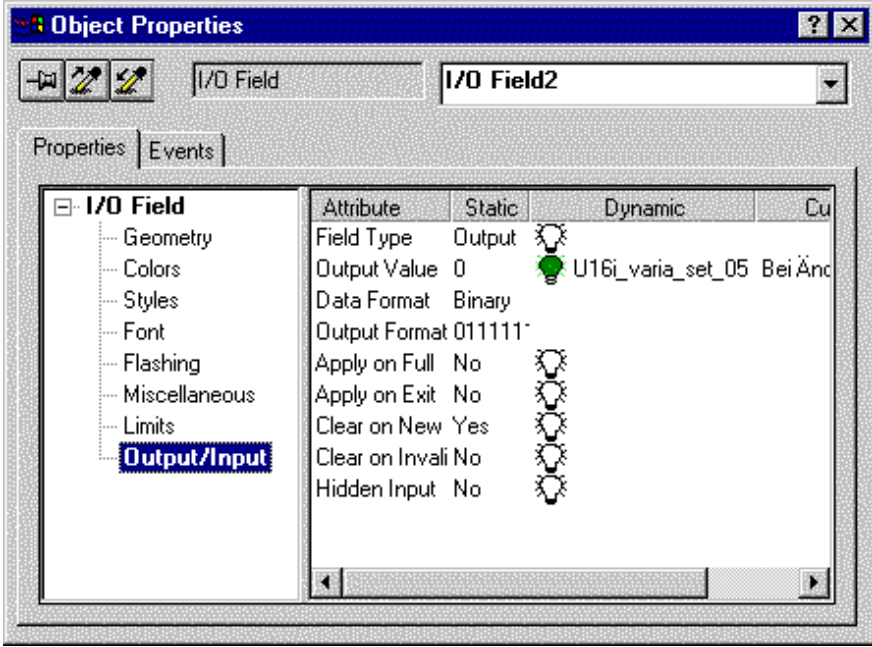
The status of a bit in a word is to be changed when this bit is selected. We want to be able to select several bits.

Implementation Concept








For the implementation of changing the bit stati, we will use a *Windows Object* → *Check-Box*. If one of the *Check-Box* fields is selected with the , the bit assigned to it is changed in the *internal tag* using a *direct connection*. To display the bit pattern, the *Smart Object* → *I/O Field* is used. The output value of the *I/O Field* is connected to an internal tag.

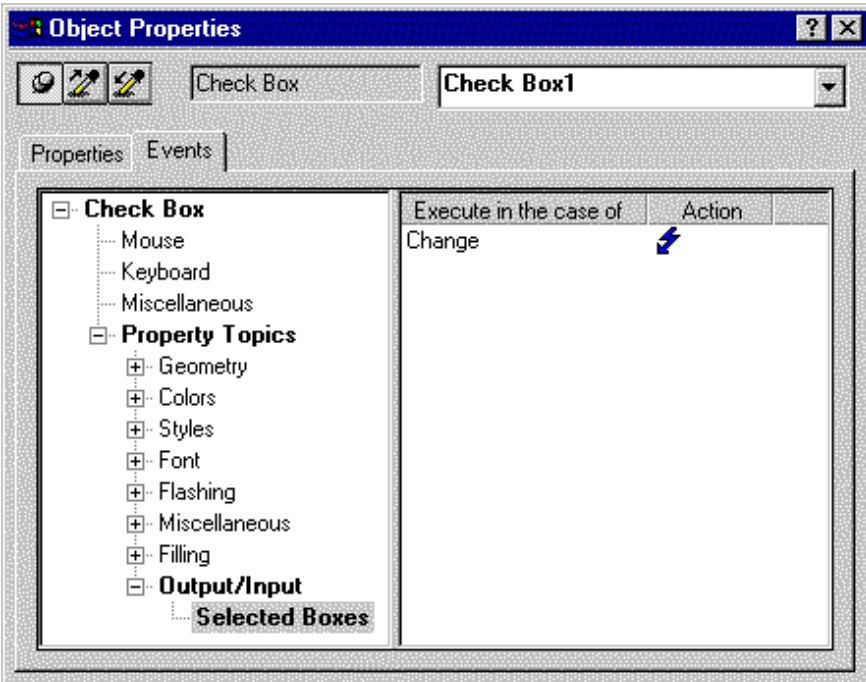
Implementation in the WinCC project

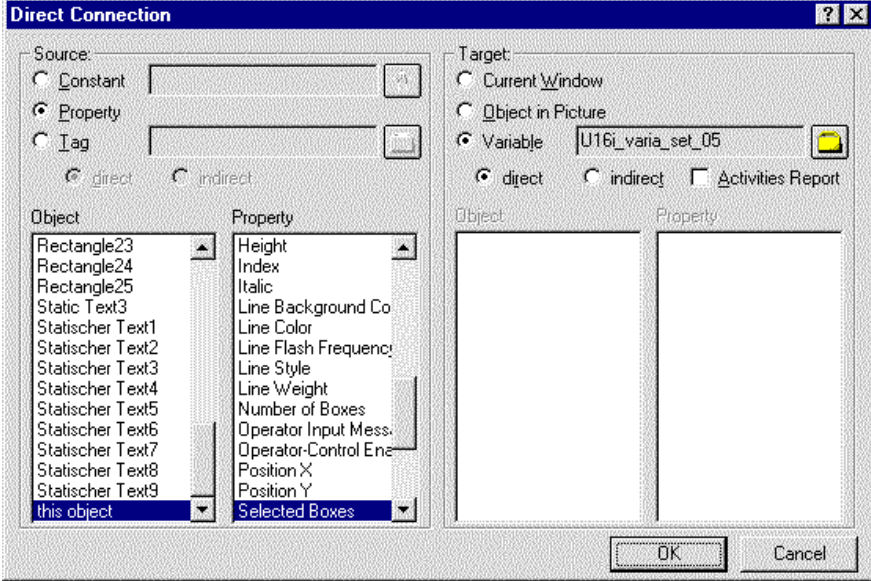
Step	Procedure: Setting a Bit directly via a Check-Box and Direct Connection
1	Create a tag of the <i>Unsigned 16-Bit Value</i> type in Tag Management. In this sample, the <i>U16i_varia_set_05</i> tag is used.
2	In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field2</i> object is used. During the creation of the <i>I/O Field</i> , set the <i>U16i_varia_set_05</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> . Via <i>Properties</i> → <i>Output/Input</i> , change the <i>Data Format</i> to <i>Binary</i> and the <i>Output Format</i> to <i>0111111111111111</i> .



The screenshot shows the 'Object Properties' dialog box for an 'I/O Field' object named 'I/O Field2'. The 'Output/Input' tab is active. The 'Field Type' is set to 'Output'. The 'Output Value' is '0'. The 'Data Format' is 'Binary'. The 'Output Format' is '0111111111111111'. The 'Update' field is set to 'Upon Change'. The 'Dynamic' column shows a green lightbulb icon next to 'U16i_varia_set_05 Bei Änd'.

Attribute	Static	Dynamic	Cu
Field Type	Output		
Output Value	0		U16i_varia_set_05 Bei Änd
Data Format	Binary		
Output Format	0111111111111111		
Apply on Full	No		
Apply on Exit	No		
Clear on New	Yes		
Clear on Invali	No		
Hidden Input	No		

Step	Procedure: Setting a Bit directly via a Check-Box and Direct Connection
3	In the same picture, configure a <i>Windows Object</i> → <i>Check-Box</i> . In this sample, the <i>Check-Box1</i> object is used. At <i>Properties</i> → <i>Geometry</i> → <i>Number of Boxes</i> , change the default value 3 to 16.
4	Select the Index 1 via <i>Properties</i> → <i>Font</i> → <i>Index</i> → 1. Enter the appropriate text for the selected index at <i>Properties</i> → <i>Font</i> → <i>Text</i> → <i>bit 0</i> . In the same manner, configure the texts for the remaining index entries.
5	<p>At <i>Event</i> → <i>Property Topics</i> → <i>Selected Boxes</i>, make this event dynamic using a <i>direct connection</i>.</p>  <p>The screenshot shows the 'Object Properties' dialog box for 'Check Box1'. The 'Events' tab is selected. On the left, a tree view shows 'Property Topics' expanded, with 'Selected Boxes' selected. On the right, the event list shows 'Change' with a lightning bolt icon, indicating a direct connection.</p>

Step	Procedure: Setting a Bit directly via a Check-Box and Direct Connection
6	<p>In the <i>Direct Connection</i> dialog, connect the source Property → <i>this object</i> → <i>Selected Boxes</i> with the target Variable → <i>U16i_varia_set_05</i>. Apply the settings by clicking on the <i>OK</i> button.</p> 
7	<p>Configure two <i>Windows Objects</i> → <i>Buttons</i>. In this sample, the <i>Button2</i> and <i>Button3</i> objects are used. These will be used to set and reset all bits.</p>
8	<p>For <i>Button2</i>, create a <i>direct connection</i> at <i>Event</i> → <i>Mouse</i> → <i>Mouse Action</i>. Connect the source <i>Constant</i> → <i>65535</i> with the target <i>Object in Picture</i> → <i>Check-Box1</i> → <i>Selected Boxes</i>. Apply the settings by clicking on the <i>OK</i> button. The constant selected corresponds to the binary number 1111111111111111.</p> <p>For <i>Button3</i>, create a <i>direct connection</i> in the same manner, but with the source <i>Constant</i> → <i>0</i>.</p>

Note for the General Application

The following adaptations must be made before the general application:

- The tags must be adapted in the *direct connections*.

2.4.2 Selecting a Bit and Changing its Status (example 01)


Task Definition

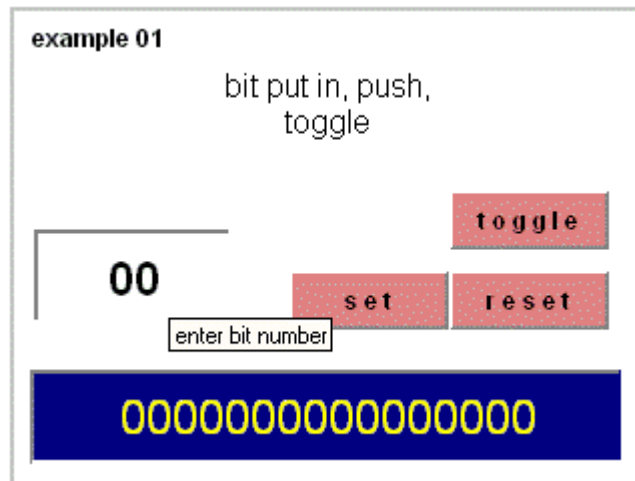
By entering a bit number and pressing a *Button*, the status of the corresponding bit in a *Word* is to be changed. It is to be switched from 0 to 1 and vice versa.

Implementation Concept

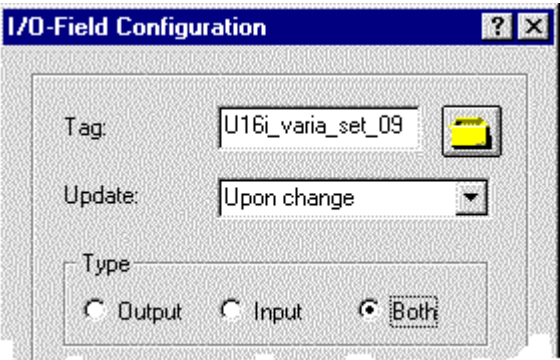
For the implementation of the change of the bit status, a Windows Object → *Button* is used.

To enter the bit number and display the bit pattern, a Smart Object → *I/O Field* is used.

When entering the bit number and pressing the *Button* using the , the selected bit in an internal tag is changed. The change is realized using a *C-Action*.



Implementation in the WinCC project

Step	Procedure: Changing Bits in a Word
1	Create two tags of the <i>Unsigned 16-Bit Value</i> type in Tag Management. In this sample, the <i>U16i_varia_set_08</i> and <i>U16i_varia_set_09</i> tags are used.
2	<p>In a picture, configure the <i>Smart Object</i> → <i>I/O Field</i>. In this sample, the <i>I/O Field2</i> object is used. During its configuration, connect the <i>I/O Field</i> to the <i>U16i_varia_set_09</i> tag. Change the update default value from 2 s to <i>Upon Change</i>. The bit number is entered in this input field.</p> 
3	For the display of the bit stati, configure a second <i>I/O Field</i> . In this sample, the <i>I/O Field1</i> object is used. During its configuration, connect the <i>I/O Field</i> to the <i>U16i_varia_set_08</i> tag. Change the update default value from 2 s to <i>Upon Change</i> . Change the field type to <i>Output</i> . Via <i>Properties</i> → <i>Output/Input</i> , change the <i>Data Format</i> to <i>Binary</i> and the <i>Output Format</i> to <i>0111111111111111</i> .
4	In the same picture, configure three <i>Windows Objects</i> → <i>Buttons</i> . In this sample, the <i>Button1</i> , <i>Button2</i> and <i>Button3</i> objects are used.
5	For <i>Button1</i> , create a <i>C-Action</i> at <i>Event</i> → <i>Mouse</i> → <i>Press Left</i> . This <i>C-Action</i> sets the bit selected form the <i>I/O Field</i> in an <i>internal tag</i> . In the same manner, create additional <i>C-Actions</i> for the other <i>Buttons</i> to reset and toggle the bit.

C-Action at Button set

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word = (WORD)(word|1<<pos);

SetTagWord("U16i_varia_set_08",word);
}
```

- Declare the *C variables*.
- Use the *internal function GetTagWord* to read the bit position entered and the current value of the variable.
- The bit shift function (
- Assign the new value to the *internal tag*.

C-Action at Button reset

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word=(WORD)(word&^(1<<pos));

SetTagWord("U16i_varia_set_08",word);
}
```

- Declare the *C variables*.
- Use the *internal function GetTagWord* to read the bit position entered and the current value of the variable.
- The bit shift function (
- Assign the new value to the *internal tag*.

C-Action at Button toggle

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
WORD word, pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word = (WORD)(word^1<<pos);

SetTagWord("U16i_varia_set_08", word);
}
```


- Declare the *C variables*.
- Use the *internal function GetTagWord* to read the bit position entered and the current value of the variable.
- The bit shift function (
- Assign the new value to the *internal tag*.

2.4.3 The remaining Samples of this Topic

example 02

The functionality of this sample is similar to that of sample *example 01*. The basic difference is in the way the bit to be switched is selected. In this example, the bit is switched by selecting an object which represents this bit. Each object is able to read out which bit it is responsible for from its object name.

example 04


The functionality of this sample is similar to that of sample *example 02*. The difference is that the bit is immediately toggled after being selected with the . Here too, the objects are assigned to the bits via the object name.

example 05

The functionality of this sample is similar to that of sample *example 06*. The difference here is that an option group (radio-button) is used. The application of this object type means that only one bit can be set in each data word.

2.5 Indirect Addressing of Tags

Indirect Address

The solutions pertaining to this topic are accessed in the Project_TagHandling project by selecting the Button displayed above using the . The samples are configured in the varia_3_chapter_04.pdl picture.

2.5.1 Indirect Addressing via a Direct Connection (example 01)

Task Definition

In an *I/O Field*, various process values are to be displayed. The corresponding values are to be selected via *Buttons*.

Implementation Concept

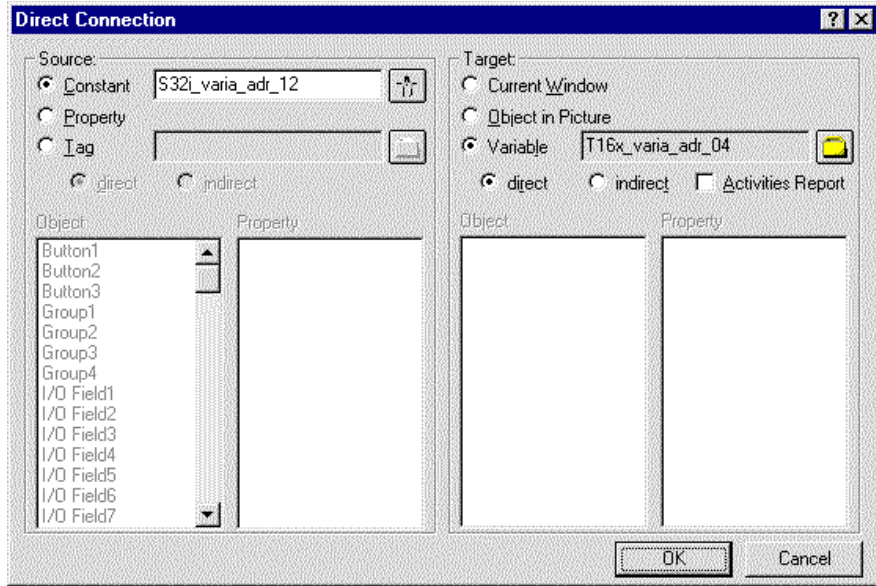
For the implementation of selecting the corresponding process values, we will use a *Windows Object* → *Button*.

For the display of the process values, we will use a *Smart Object* → *I/O Field* and the indirect addressing option in WinCC. Three additional *Smart Objects* → *I/O Fields* are created to permit the direct entry of the process values.

Implementation in the WinCC Project

Step	Procedure: Indirect Addressing via a Direct Connection
1	Create three tags of the <i>Signed 32-Bit Value</i> in Tag Management. In this sample, the <i>S32i_varia_adr_12</i> , <i>S32i_varia_adr_13</i> and <i>S32i_varia_adr_14</i> tags are used. These contain the process values to be displayed.
2	Create a tag of the <i>Text Tag 16-Bit Character Set</i> type in Tag Management. In this sample, the <i>T16x_varia_adr_04</i> tag is used. This tag will be used as an address tag.
3	In a picture, configure a <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field4</i> object is used. During the creation of the <i>I/O Field</i> , set the <i>T16x_varia_adr_04</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> . At <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i> , activate the check-box in the <i>Indirect</i> column.

Attribute	Static	Dynamic	Cu
Field Type	Output	<input type="checkbox"/>	
Output Value	56,0000	<input checked="" type="checkbox"/> T16x_varia_adr_04 Bei Änc	
Data Format	Decimal		
Output Format	09999		
Apply on Full	No	<input type="checkbox"/>	
Apply on Exit	No	<input type="checkbox"/>	
Clear on New	Yes	<input type="checkbox"/>	
Clear on Invali	No	<input type="checkbox"/>	
Hidden Input	No	<input type="checkbox"/>	

Step	Procedure: Indirect Addressing via a Direct Connection
4	In the same picture, configure 3 additional <i>I/O Fields</i> . In this sample, the <i>IO-Field1</i> to <i>IO-Field3</i> objects are used. During the creation of <i>IO-Field1</i> , set the <i>S32i_varia_adr_12</i> tag and the trigger <i>Upon Change</i> in the <i>configuration dialog</i> . In the same manner, configure the die <i>I/O Fields 2 and 3</i> , but connect each to a different address tag.
5	Configure an object of the <i>Standard Object</i> → <i>Static Text</i> type. In this sample, the <i>Static Text1</i> object is used. This object indicates which process value is currently being displayed. The text in the object is automatically supplied by the <i>Button</i> .
6	In the same picture, configure three <i>Windows Objects</i> → <i>Buttons</i> . In this sample, the <i>Button1</i> , <i>Button2</i> and <i>Button3</i> objects are used.
7	<p>For <i>Button1</i>, configure a <i>direct connection</i> at <i>Event</i> → <i>Mouse</i> → <i>Press Left</i>. Connect the <i>source Constant</i> → <i>S32i_varia_adr_12</i> with the <i>target Variable</i> → <i>T16x_varia_adr_04</i>. Apply the settings by clicking on the <i>OK</i> button.</p> 
8	Create another <i>direct connection</i> at <i>Event</i> → <i>Mouse</i> → <i>Mouse Action</i> . Connect the <i>source Property</i> → <i>this object</i> → <i>Text</i> with the <i>target Object in Picture</i> → <i>Static Text1</i> → <i>Text</i> . Apply the settings by clicking on the <i>OK</i> button.
9	Configure <i>Button2</i> and <i>Button3</i> in the same manner as <i>Button1</i> . For the first <i>direct connection</i> , the tag name of the <i>source</i> must be changed. The second <i>direct connection</i> can be applied without any changes.

Note for the General Application

The following adaptations must be made for the general application:

- The tag names must be adapted.

2.5.2 Multiplex Display with Indirect Addressing and C-Action (example 02)

Task Definition

Three different process values of a container are to be displayed. The same display is, however, set up for several containers. The relevant process values are displayed by selecting the corresponding container.

Implementation Concept

For the implementation of selecting the corresponding container, a *Windows Object* → *Option Group* is used.

For the display of the process values, the *Smart Objects* → *I/O Fields* and the indirect addressing option in WinCC are used.

The containers with the corresponding values are displayed in *example 04*.

Implementation in the WinCC project

Step	Procedure: Multiplex Display with Indirect Addressing
1	Create nine tags of the <i>Signed 32-Bit Value</i> type in Tag Management. In this sample, the <i>S32i_varia_adr_03</i> to <i>S32i_varia_adr_11</i> tags are created. These tags contain the corresponding process values of the containers.
2	Create three tags of the <i>Text Tag 16-Bit Character Set</i> type in Tag Management. In this sample, the <i>T16x_varia_adr_01</i> , <i>T16x_varia_adr_02</i> and <i>T16x_varia_adr_03</i> tags are used. These tags will be used as address tags for the <i>I/O Fields</i> .
3	Configure three <i>Smart Objects</i> → <i>I/O Fields</i> . In this sample, the <i>I/O Field5</i> , <i>I/O Field6</i> and <i>I/O Field7</i> objects are used.
4	During the creation of <i>I/O Field5</i> , the <i>T16x_varia_adr_01</i> tag is set in the <i>configuration dialog</i> . Change the <i>Update</i> to <i>Upon Change</i> and the <i>Field Type</i> to <i>Output</i> . At <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i> , activate the checkbox in the <i>Indirect</i> column.
5	In the same manner, configure the remaining <i>I/O Fields</i> , but connect each to a different address tag.
6	Configure a <i>Windows Object</i> → <i>Option Group</i> . In this sample, the <i>Option Group1</i> object is used.
7	Select the Index 1 via <i>Properties</i> → <i>Font</i> → <i>Index</i> . Enter the appropriate text for the selected index at <i>Properties</i> → <i>Font</i> → <i>Text</i> → <i>Container1</i> . In the same manner, configure the texts for the remaining index values.
8	At <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Selected Boxes</i> , create a <i>C-Action</i> . This action writes to the address tag depending on the field selected.

C-Action at the Option Group

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l
{
char address1[20], address2[20], address3[20];
switch(value) {
case 2: {
strcpy(address1, "S32i_varia_adr_03");
strcpy(address2, "S32i_varia_adr_06");
strcpy(address3, "S32i_varia_adr_09");
} //case 2
break;
case 4: {
strcpy(address1, "S32i_varia_adr_04");
strcpy(address2, "S32i_varia_adr_07");
strcpy(address3, "S32i_varia_adr_10");
} //case 4
break;
default: {
strcpy(address1, "S32i_varia_adr_05");
strcpy(address2, "S32i_varia_adr_08");
strcpy(address3, "S32i_varia_adr_11");
} //default
break;
} //switch

SetTagChar("T16x_varia_adr_01", address1);
SetTagChar("T16x_varia_adr_02", address2);
SetTagChar("T16x_varia_adr_03", address3);
}
```

- Declaration of three *C variables* as an array of characters.
- Copy the variable names according to the input status into the tags declared previously. The input status is stored in the predefined *value* variable.
- Assign the corresponding variable names to the address variables.

Note for the General Application

The following adaptations must be made before the general application:

- The variable names must be adapted.

2.5.3 Indirect Addressing with C-Action (example 03)

Task Definition

In an *I/O Field*, various process values are to be displayed. The corresponding values are to be selected via an *Option Group*.

Implementation Concept

For the implementation of selecting the corresponding process values, a *Windows Object* → *Option Group* is used. For the display, a *Smart Object* → *I/O Field* and the indirect addressing option in WinCC is used.

Implementation in the WinCC project

Step	Procedure: Indirect Addressing with C-Action
1	Create three tags of the <i>Signed 32-Bit Value</i> in Tag Management. In this sample, the <i>S32i_varia_adr_00</i> , <i>S32i_varia_adr_01</i> and <i>S32i_varia_adr_02</i> tags are used. These contain the process values to be displayed.
2	Create a tag of the <i>Text Tag 16-Bit Character Set</i> type in Tag Management. In this sample, the <i>T16x_varia_adr_00</i> tag is used. This tag will be used as an address tag.
3	In a picture, configure a <i>Smart Object</i> → <i>I/O Field</i> . In this sample, the <i>I/O Field8</i> object is used. During the creation of the <i>I/O Field</i> , set the <i>T16x_varia_adr_00</i> tag in the <i>configuration dialog</i> . Change the 2 s default value in the <i>Update</i> field to <i>Upon Change</i> and set the <i>Field Type</i> to <i>Output</i> . At <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i> , activate the check-box in the <i>Indirect</i> column.
4	In the same picture, configure a <i>Windows Object</i> → <i>Option Group</i> . In this sample, the <i>Option Group2</i> object is used.
5	Select the Index 1 via <i>Properties</i> → <i>Font</i> → <i>Index</i> . Enter the appropriate text for the selected index at <i>Properties</i> → <i>Font</i> → <i>Text</i> → <i>Fill Level</i> . In the same way, configure the texts for the remaining index values.
6	At <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Selected Boxes</i> , create a <i>C-Action</i> . This action writes to the address tag depending on the field selected.

C-Action at the Option Group

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l
{
char address[40];

//set tag according to input value
switch(value) {
case 2: strcpy(address, "S32i_varia_adr_01");
break;
case 4: strcpy(address, "S32i_varia_adr_02");
break;
default: strcpy(address, "S32i_varia_adr_00");
} //switch
SetTagChar("T16x_varia_adr_00", address);
}
```

- Assign variable names to the *T16x_varia_adr_00* address variable according to the input status. The input status is stored in the predefined *value* tag.

Note for the General Application

The following adaptations must be made before the general application:

- The variable names must be adapted.


2.5.4 The remaining Samples of this Topic

example 04

The functionality of this example is to display the process values used in *example 02*.

2.6 Simulation of Tags

An orange rectangular button with the word "Simulation" in white text.

The solutions pertaining to this topic are accessed in the *Project_TagHandling* project by selecting the *Button* displayed above using the . The samples are configured in the *varia_3_chapter_05.pdl* picture.

Definition

The term simulation refers to changing the contents of a tag without a process driver connection. The simulation is performed using utility programs.

2.6.1 Simulation of a Triangular Oscillation via a C-Action (example 01)

Task Definition

A triangular oscillation simulation with setable values for the maximum value and minimum value is to be created. The plausibility of these values is to be verified as they are entered. The simulation is to be started and stopped via a *Button*. Another *Button* is used to reset the tag value to zero.

Implementation Concept

For the implementation of starting/stopping and initializing the simulation, two Windows Objects → Buttons are used. To display the tag value and to input the maximum and minimum values, Smart Objects → I/O Fields are used. If the simulation is started while the maximum and minimum values set are identical, a message box will be displayed.

Implementation in the WinCC Project

Step	Procedure: Simulation of a Triangular Oscillation via a C-Action
1	Create three tags of the <i>Signed 32-Bit Value</i> in Tag Management. In this sample, the <i>S32i_varia_sim_00</i> , <i>S32i_varia_sim_02</i> and <i>S32i_varia_sim_03</i> tags are used.
2	Create two tags of the <i>Binary Tag</i> type. In this sample, the <i>BINi_varia_sim_01</i> and <i>BINi_varia_sim_04</i> tags are used.
3	Configure three <i>Smart Objects</i> → <i>I/O Fields</i> . In this sample, the <i>I/O Field1</i> , <i>I/O Field2</i> and <i>I/O Field3</i> objects are used.
4	During the creation of <i>I/O Field1</i> , set the tag <i>S32i_varia_sim_03</i> and the trigger <i>Upon Change</i> in the <i>configuration dialog</i> . At <i>Properties</i> → <i>Output Format</i> , change the format to <i>0999</i> . In the same manner, configure the <i>I/O Field2</i> , but set the tag <i>S32i_varia_sim_02</i> .
5	To check the plausibility of the <i>I/O Field1</i> object, configure a <i>Tag Connection</i> to the <i>S32i_varia_sim_02</i> tag at <i>Properties</i> → <i>Limits</i> → <i>High Limit Value</i> . In the same manner, configure the <i>S32i_varia_sim_03</i> tag as the <i>Low Limit Value</i> of <i>I/O Field2</i> .
6	During the creation of the <i>I/O Field3</i> object, set the tag <i>S32i_varia_sim_00</i> , the trigger <i>Upon Change</i> and the <i>Field Type</i> to <i>Output</i> in the <i>configuration dialog</i> . At <i>Properties</i> → <i>Output/Input</i> → <i>Output Format</i> , change the format to <i>0999</i> .
7	Configure a <i>Smart Object</i> → <i>Picture Window</i> , in this sample this is the <i>dialog box</i> . Via <i>Properties</i> → <i>Miscellaneous</i> , change the properties <i>Moveable</i> and <i>Border</i> to <i>Yes</i> and the <i>Picture Name</i> to <i>varia_5_window_00</i> . This picture can be taken from the sample project for use in your own projects; the info text and the title can be changed to suit your needs.
8	Configure a <i>Windows Object</i> → <i>Button</i> , in this sample this is the <i>Button2</i> . At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create a <i>direct connection</i> . Connect the <i>source Constant</i> → <i>1</i> with the <i>target variable</i> → <i>BINi_varia_sim_04</i> . This <i>Button</i> is used for the initialization.

Step	Procedure: Simulation of a Triangular Oscillation via a C-Action
9	Configure another <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button1</i> is used. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create a <i>C-Action</i> that negates the status of the <i>BINi_varia_sim_01</i> tag. At <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> , create a <i>C-Action</i> that executes the tag simulation.
10	To display the simulation status, configure a <i>Smart Object</i> → <i>Status Display</i> . In this sample, the <i>Status Display1</i> is used. In the <i>configuration dialog</i> , set the tag <i>BINi_varia_sim_01</i> and trigger <i>Upon Change</i> . Add another status. For the status <i>0</i> , set the picture <i>glühbirne_2_24Bit.gif</i> and for status <i>1</i> the picture <i>glühbirne_1_24Bit.gif</i> .

C-Action for the Tag Simulation

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    BOOL state;
    static DWORD lowstore = 0;
    static BOOL statestore = 0;
    static DWORD r = 1;
    static DWORD i = 0;
    static DWORD box = 0;
    int high, low;

    //if button init was pressed
    if (GetTagBit("BINi_varia_sim_04")) {
        (i=lowstore);
        (r=1);
        SetTagDWord("S32i_varia_sim_00", i);
        SetTagDWord("BINi_varia_sim_04", 0);
    }

    //get simulator state
    state=GetTagBit("BINi_varia_sim_01");

    if (state!=statestore) (box=0);

    statestore=state;

    //get limits
    high=GetTagDWord("S32i_varia_sim_02");
    low=GetTagDWord("S32i_varia_sim_03");

    //set low limit store
    if (low!=lowstore){
        lowstore = low;
        i=low;
    }//if

    //if limits different
    if (high!=low) {
        box=0;
        //if simulator is activated
        if (state==TRUE) {
            //inc or dec according to direction
            if (r==1) (i=i+1);
            else (i=i-1);
            //set direction
            if (i==high) (r=0);
            if (i==low) (r=1);
            //init simulator if limit overflow
            if ((i>high)||i<low){
                (i=low);
                (r=1);
            }//if
            //set new value
            SetTagDWord("S32i_varia_sim_00", i);
        }//if state
    }//if (high!=low)
    //set visible message box
    if ((high==low)&&(state==1)&&(box==0)){
        box++;
        SetVisible("varia_3_chapter_05.PDL", "dialog box", 1);
    }
    return 80; //x-pos
}
```

- Declare the tags.
- If *Button2* (init) is pressed, set the tag value memory to the stored minimum value, the counter direction to ascending, the value of the *internal tag S32i_varia_sim_00* to the stored minimum value and turn the simulator off.
- Read in the simulator status.
- If the status has changed, the message box is allowed to pop up.
- Save the status.
- Read in the maximum and minimum values.
- If the minimum value changes, update the minimum value memory.
- If the maximum and minimum values are different, the message box is allowed to pop up and the simulation is performed if the simulator is turned on. Counting up or counting down according to the direction tag; if the limit values are reached, the direction is reversed; if the limit values are exceeded, initialization takes place and *S32i_varia_sim_00* tag is set to the minimum value.
- If the simulator is turned on, the message box display enabled and the maximum and minimum values agree, the message box is set to visible.
- The return value is the X position of the *Button1* object.

2.6.2 Simulation via an External Program (example 02)

WinCC provides its own simulation program, which can simulate tags using a number of different methods. This simulation program must be installed using the Setup.exe program in the located in the folder SmartTools → CC_Simulator on the WinCC CD-ROM.

Task Definition

Tags are to be simulated using the WinCC tag simulator.

Implementation Concept

For the implementation, will use a number of tags - which will be displayed in *Smart Objects* → *I/O Fields* - whose contents will be controlled by the tag simulator.

Implementation in the WinCC Project

Step	Procedure: Simulation via an External Program
1	Create two <i>internal tags</i> of the <i>Signed 32-Bit Value</i> type in Tag Management. In this sample, the <i>S32i_varia_sim_05</i> and <i>S32i_varia_sim_06</i> tags are used.
2	Configure two objects of the <i>Smart Objects</i> → <i>I/O Fields</i> type. In this sample, the <i>I/O Field4</i> and <i>I/O Field5</i> objects are used.
3	During the creation of <i>I/O Field4</i> , set the tag <i>S32i_varia_sim_05</i> , the trigger to upon change and the field type to output in the configuration dialog. At <i>Properties</i> → <i>Output/Input</i> , change the <i>Output Format</i> to <i>0999.999</i> . In the same manner, make the settings for <i>I/O Field5</i> , but set the <i>S32i_varia_sim_06</i> tag.
4	Start the tag simulator by clicking on the <i>Simulator</i> button. The tag simulator is started via a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . If the tag simulator has not been installed at the default location, set the correct path via the <i>Path</i> button. If the simulation program is started using a method other than the one described, you must ensure that the project in question is in runtime mode.
5	In the simulator displayed, select the <i>S32i_varia_sim_05</i> tag from Tag Management via the <i>Edit</i> → <i>New Tag</i> menus. Select the <i>Inc</i> tab and enter <i>Start Value</i> and <i>End Value</i> . In this sample, the values <i>0</i> and <i>20</i> have been used. The simulation is started by selecting the <i>Active</i> menu point. The value of the tag is increased from <i>0</i> to <i>20</i> , after which the simulation is restarted from <i>0</i> .
6	Proceed in the same manner with the <i>S32i_varia_sim_06</i> tag. In this sample, the <i>Sinus</i> tab has been selected: The <i>Amplitude</i> has been set to <i>50</i> , the <i>Offset</i> to <i>50</i> and the <i>Oscillation Time</i> to <i>25</i> .

C-Action at the Simulator Button

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{
char simdeupath[200];
char program[15];
int result;

if (GetTagBit("BINi_varia_sim_10")) strcpy(program, "\\simeng.exe");
else strcpy(program, "\\simdeu.exe");

//build path to simdeu
strcpy(simdeupath, GetTagChar("T16x_varia_sim_07"));
strcat(simdeupath, program);

//execute simdeu
result=ProgramExecute(simdeupath);

//if not able to execute simdeu set visible message box
if (result<32) {
    SetVisible("varia_3_chapter_05.PDL", "error message", 1);
}
}
```

- Declare the tags.
- If the BINi_varia_sim_10 variable is set, the name of the English simulator program is written to the program variable. Otherwise the name of the German simulator is written to the program variable.
- Via the internal function GetTagChar, the path of the simulator program is read.
- Add the start file to the path.
- Start the simulator.
- If the path specification is incorrect, output an error message.

Note:

In the sample project, the flag icon can be selected to start either the English or German version of the tag simulator.


Note for the General Application

The following adaptations must be made before the general application:

- The tags to be simulated and the method of simulation must be customized to suit your own requirements.

2.7 Importing / Exporting Tags



The solutions pertaining to this topic are accessed in the *Project_TagHandling* project by selecting the *Button* displayed above using the . The samples are configured in the *varia_3_chapter_06.pdl* picture.

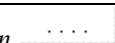

Task Definition

The contents of Tag Management are to be read by an utility program and edited in MS Excel (spreadsheet program). The modified data is to be able to be imported back into the WinCC project again. This procedure makes it possible to create a large number of tags without any great effort.

Implementation Concept

For the implementation in the project, two *Windows Objects* → *Buttons* are used to start the *var_imex.exe* import/export program and the *excel.exe* program. The path to each of these programs can be set via two *Smart Objects* → *I/O Fields*.

Implementation in the WinCC Project

Step	Procedure: Importing / Exporting Tags
1	Set the correct paths to the <i>excel.exe</i> and <i>var_imex.exe</i> programs.
2	Start the <i>var_exim.exe</i> program by clicking on the <i>Button Imp/Exp</i> during runtime. The program can also be started directly from the Windows Explorer, without runtime having to be active.
3	Via the <i>Button</i>  , set the path to the <i>Project_TagHandling</i> project and select the <i>Project_TagHandling.mcp</i> file.
4	Select the <i>Export</i> selection field. The C-Action for calling external programs is described in sample 1.6.2. Then click on <i>Execute</i> → <i>OK</i> . The export of the tags is now performed. The program generates a file with the extension <i>vex</i> containing information about the tags, a second file with the extension <i>cex</i> containing information about the connections to the PLC and a third file with the extension <i>dex</i> containing information about tags of the <i>Data Structure</i> type.
5	Start <i>Excel</i> and open the <i>Project_vex.csv</i> file just generated via <i>File</i> → <i>Open</i> .
6	To configure 100 tags of the unsigned 16-Bit value type, proceed as follows. <i>The names assigned to these tags are ranging from U16i_varia_impex_00 to U16i_varia_impex_99.</i>
7	In the first empty line of the first column enter the name <i>U16i_varia_impex_00</i> . Select the cell and move the mouse pointer to the bottom right corner. While keeping the  pressed, drag the mouse pointer downward to automatically fill in the remaining 99 cells.


Step	Procedure: Importing / Exporting Tags
8	In the second column, enter a *; in the third column, enter <i>Internal Tag</i> ; in the fourth column, enter <i>impexp</i> as the group name; in the fifth column, enter 2 and in the sixth column, enter 5 as the code for an <i>Unsigned 16-Bit Value</i> . In the remaining columns, the value 0 is entered. Fill in the remaining 99 lines automatically.
9	Open <i>Var_imex.exe</i> again via the task bar and select the <i>Import</i> selection field. Then click on <i>Execute</i> → <i>OK</i> . After completing the import of the tags, exit the program.
10	100 new tags have now been created in Tag Management.

Note:

Runtime does not have to be active to while importing and exporting tags.

2.8 Using Structure Tags



The solutions pertaining to this topic are accessed in the *Project_TagHandling* project by selecting the *Button* displayed above using the . The samples are configured in the *varia_3_chapter_07.pdl* picture.

Definition

This data type enables you to generate a data structure that forms a logical unit. Structure tags consist of various default data types.

2.8.1 Controlling a Valve with a Structure Tag (example 01)



Task Definition



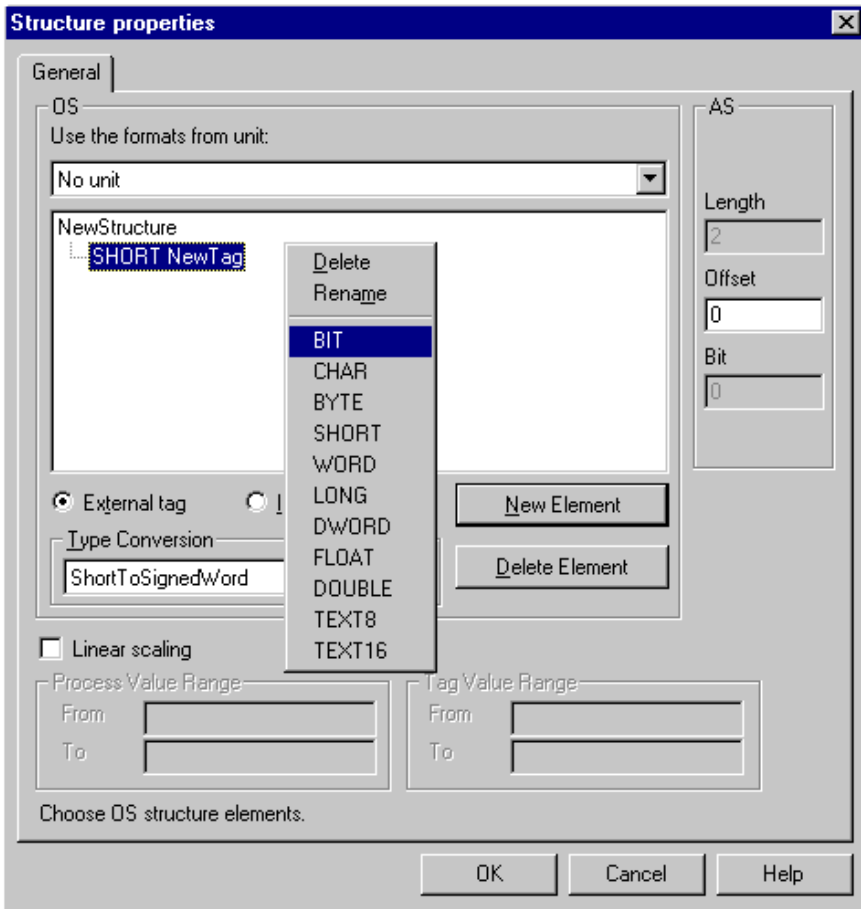












Different stati of a valve are to be displayed with the aid of a structure tag.

Implementation Concept

For the implementation, we use two *Windows Objects* → *Buttons*, with which the valve is turned on and off and a fault condition is simulated. To display the valve, we use *Standard Objects* → *Polygons*.

Implementation in the WinCC Project

Step	Procedure: Controlling a Valve with a Structure Tag
1	<p>Define a new structure tag in the <i>WinCC Explorer</i>. At <i>Data Types</i>, select → <i>Structure Types</i> via a  and then <i>New Structure</i> from the pop-up menu.</p> 

Step	Procedure: Controlling a Valve with a Structure Tag																				
2	<p>In the following window,  on <i>New Structure</i> and then select <i>Rename</i> from the pop-up menu. In this sample, the name <i>valve</i> is used. Via the <i>New Element</i> button, add a new structure element. Via a  on the just created element, set its data type to <i>Bit</i>.</p> 																				
3	<p>Via the <i>Rename</i> button, change the element name to <i>aktivated</i> and select the <i>internal</i> radio-button. Define further structure elements as follows:</p> <pre data-bbox="535 1449 714 1606"> valve: ├── BIT aktivated ├── BIT open ├── BIT closed └── BIT error </pre>																				
4	<p>In Tag Management, create a tag of the <i>valve</i> type. In the sample, the <i>STUi_varia_str_00</i> tag is used. This action creates the following <i>Binary Tags</i>.</p> <table border="1" data-bbox="532 1732 1393 1890"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Parameters</th> <th>Last change</th> </tr> </thead> <tbody> <tr> <td> STUi_varia_str_00.aktivated</td> <td>Binary Tag</td> <td>Internal tag</td> <td>07/10/97 02:26:14</td> </tr> <tr> <td> STUi_varia_str_00.open</td> <td>Binary Tag</td> <td>Internal tag</td> <td>07/10/97 02:26:14</td> </tr> <tr> <td> STUi_varia_str_00.closed</td> <td>Binary Tag</td> <td>Internal tag</td> <td>08/21/97 03:45:32</td> </tr> <tr> <td> STUi_varia_str_00.error</td> <td>Binary Tag</td> <td>Internal tag</td> <td>07/10/97 02:26:14</td> </tr> </tbody> </table>	Name	Type	Parameters	Last change	 STUi_varia_str_00.aktivated	Binary Tag	Internal tag	07/10/97 02:26:14	 STUi_varia_str_00.open	Binary Tag	Internal tag	07/10/97 02:26:14	 STUi_varia_str_00.closed	Binary Tag	Internal tag	08/21/97 03:45:32	 STUi_varia_str_00.error	Binary Tag	Internal tag	07/10/97 02:26:14
Name	Type	Parameters	Last change																		
 STUi_varia_str_00.aktivated	Binary Tag	Internal tag	07/10/97 02:26:14																		
 STUi_varia_str_00.open	Binary Tag	Internal tag	07/10/97 02:26:14																		
 STUi_varia_str_00.closed	Binary Tag	Internal tag	08/21/97 03:45:32																		
 STUi_varia_str_00.error	Binary Tag	Internal tag	07/10/97 02:26:14																		

Step	Procedure: Controlling a Valve with a Structure Tag
5	Configure two <i>Windows Objects</i> → <i>Buttons</i> , in this sample, the <i>Button1</i> and <i>Button2</i> objects are used. At <i>Button1</i> → <i>Event</i> → <i>Mouse Action</i> → <i>Press Left</i> , create a <i>C-Action</i> that turns the valve on or off. In the same manner, create a <i>C-Action</i> at <i>Button2</i> that turns the error bit on or off.
6	For <i>Button1</i> , create a <i>C-Action</i> at <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> that simulates the external processes at the valve.
7	We then create three different pictures to display the on, off and error stati of the valve. In the sample, these pictures contain two <i>Standard Objects</i> → <i>Polygons</i> . They are positioned one on top of the other and shown or hidden depending on the status of the valve.

Note for the General Application

The following adaptations must be made before the general application:

- The structure type name and its comprising structure element types and structure element names must be adapted.
- *C-Actions* for the simulation of the external processes at the valve are not necessary for actual applications.

3 Picture Configuration (Project_CreatePicture)

The project created in this chapter can also be copied directly from the online document to your hard drive. By default, it will be stored to the C:\Configuration_Manual folder.



Project_CreatePicture

Project Details

This project presents various ways of structuring and opening pictures in WinCC. Picture structure and picture opening depend on two factors: on the hardware used (an industry PC in the form of an operator panel with integrated keyboard -OP47- or a PC in the control room with mouse and standard keyboard) and on the application. A manufacturer of machinery requires different things from an HMI system compared with, for example, a chemicals company.

What Possibilities does WinCC offer?

WinCC supports all screen resolutions that are supported by Windows (e.g. 640x480, 800x600, 1024x768, 1280x1024). Sometimes, overview pictures have to be displayed on a larger base (e.g. 1600x1028, 2000x1500, etc.).

WinCC allows you to create pictures with a maximum resolution of 4096 x 4096 pixels. If these dimensions are greater than the maximum resolution of the graphics system being used (video card with monitor), these pictures can be moved round using scroll bars.

Assumption

The resolution of the graphics system being used is assumed to be 1024 x 768 pixels. This resolution corresponds to the recommendations with regard to ergonomics for a graphics system with a 17" monitor.

The samples for this topic are configured in the *Project_CreatePicture* WinCC project.



Note:

The password for the Login is pictu_00. Simply click on *Login* on the window title, enter the password in the *Password* entry field and then confirm your entry.

3.1 Screen Layout and Picture Change

This chapter shows you a number of different ways of structuring and opening pictures. The basic elements (start picture, overview section and buttons section) of the screen layout are also used in the other projects.

3.1.1 Screen Layout

Task Definition

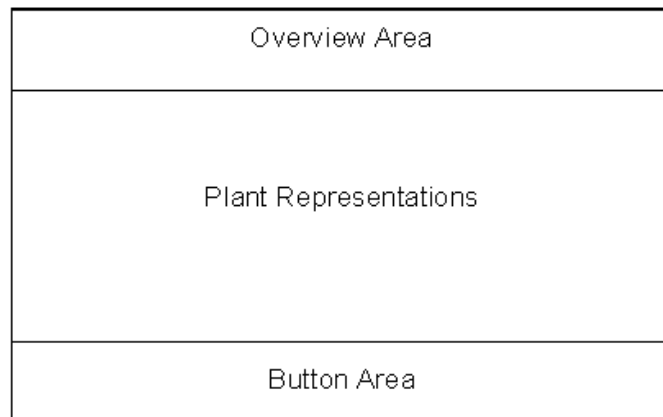
Dynamic Button Set and Overview Section

The screen is to be divided into three sections: an overview section, a buttons section, and a section for the plant pictures. The overview and buttons sections are to be adjustable. The system is located in a control room and is controlled using a mouse and a keyboard.

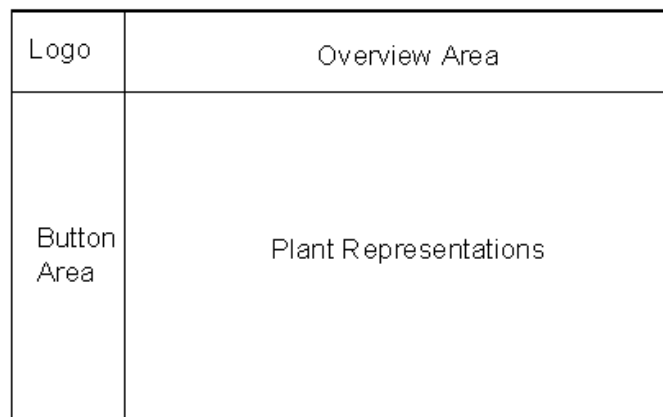
Implementation Concept

The screen is set to a resolution of 1024 x 768 pixels. We will divide the screen into three sections. We will use two different layouts for the three sections.

Layout 1



Layout 2



Layout Principle

We use a blank start picture in which we then create 3 picture windows (overview, buttons, plant). The pictures displayed in these picture windows can be swapped over during runtime as required. This gives us a solution which is very flexible and simple to modify.

Overview Section

In the overview section, we configure a logo, a picture title, a clock with the date and time, and an alarm line.

Buttons Section


In the buttons section, we configure permanent buttons which will be displayed in every picture and buttons which will be displayed depending on the plant picture displayed.

Plant Section

In the plant section, we configure the respective plant pictures.

3.2 Picture Change

Pic Change


In runtime, the samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_01.pdl* and *pictu_3_chapter_01a.pdl* pictures.

3.2.1 Opening a Picture via a Direct Connection and Displaying the Picture Name (example 01)


Task Definition



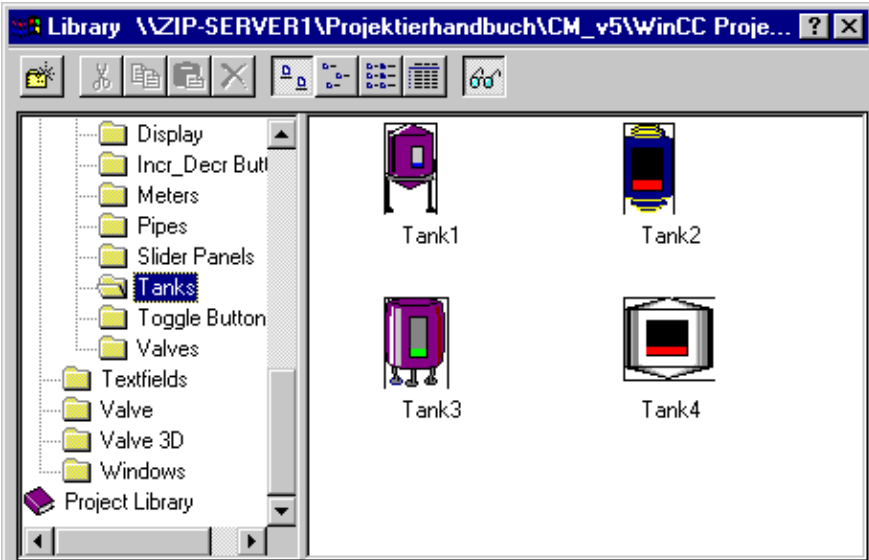
In the picture window, the picture change is to be implemented via a mouse-operated Button and the aid of a direct connection.

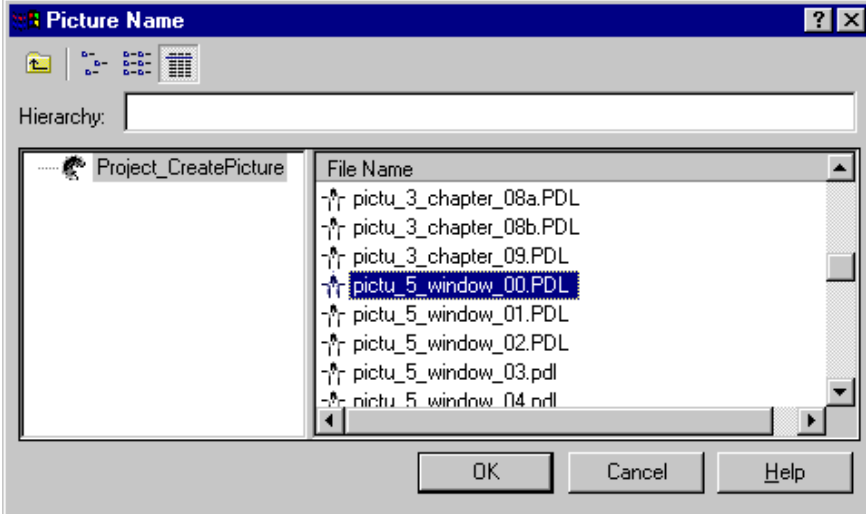

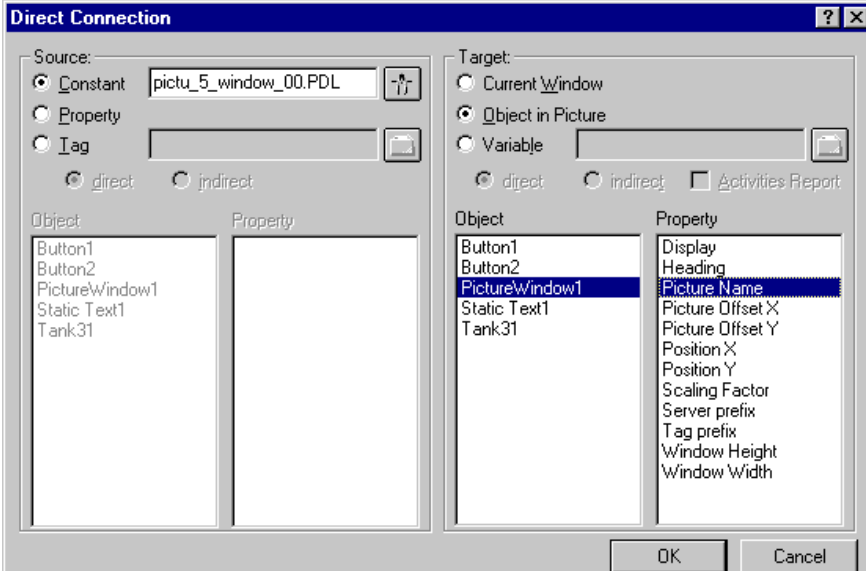
Implementation Concept


For the implementation, we will use a *Windows Object* → *Button*, which changes the picture displayed in the *Smart Object* → *Picture Window* when it is pressed with the . In the picture, the picture name is displayed using a *Standard Object* → *Static Text*.

Implementation in the WinCC Project

Step	Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name
1	Via <i>File</i> → <i>New</i> , create a new picture and via <i>File</i> → <i>Save As...</i> , save it under the name <i>pictu_5_window_00.pdl</i> . Set <i>Properties</i> → <i>Geometry</i> → <i>Width</i> to 270 and <i>Properties</i> → <i>Geometry</i> → <i>Height</i> to 280.
2	<p>In the <i>pictu_5_window_00.pdl</i> picture, configure a <i>Standard Object</i> → <i>Static Text</i>. In this sample, the <i>Static Text1</i> object is used. Set <i>Properties</i> → <i>Font</i> → <i>Bold</i> to <i>Yes</i>. At <i>Properties</i> → <i>Font</i> → <i>Text</i>, delete the default text from the <i>Static</i> column. This prevents an incorrect text from being output at the moment of the picture generation.</p> <p>Make the object at <i>Properties</i> → <i>Font</i> → <i>Text</i> dynamic using a <i>C-Action</i>. This <i>C-Action</i> returns the current picture name as the return value. As the trigger for the <i>C-Action</i>, the <i>Default Cycle</i> → <i>1 h</i> is used (low system load, no change).</p> <div data-bbox="527 1312 966 1375" style="border: 1px solid gray; padding: 2px;"> Event Name: <input type="text" value="1 h"/>  </div>

Step	Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name
3	<p>In the <i>pictu_5_window_00.pdl</i> picture, configure the information to be displayed. In the sample, the <i>tank3</i> object from the Global Library is used. The library is accessed via <i>View</i> → <i>Library</i> or via the  button on the toolbar.</p> <p>Make sure that the symbol view has been selected via the <i>Button</i>  to obtain a preview of the individual objects.</p> 
4	<p>Configure two more sample pictures for the picture change by saving the picture you have just configured via <i>File</i> → <i>Save As...</i> under the name <i>pictu_5_window_01.pdl</i> and this picture again under the name <i>pictu_5_window_02.pdl</i>. This gives us two copies of <i>pictu_5_window_00.pdl</i>. Now insert the desired content into the new pictures created. There is no need to change the <i>Static Text1</i> object for displaying the picture name.</p>
5	<p>Configure a new picture via <i>File</i> → <i>New</i>. Configure a <i>Smart Object</i> → <i>Picture Window</i> in this picture. In this sample, the <i>Picture Window1</i> object is used. Adapt the dimension of the <i>Picture Window</i> via <i>Properties</i> → <i>Geometry</i> → <i>Width</i> and <i>Properties</i> → <i>Geometry</i> → <i>Height</i> to the size of the previously created pictures. To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i>.</p>

Step	Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name
6	<p>At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the <i>pictu_5_window_00.pdl</i> picture. This sets the picture to be displayed by the <i>Picture Window1</i> object when the picture is opened.</p> 
7	<p>In the same picture, configure a <i>Windows Object</i> → <i>Button</i>. In this sample, this is the <i>Button1</i> object. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, create a direct connection.</p> <p>Select <i>Constant</i> as the <i>Source</i> and click on the now enabled button  to display a selection list of all available pictures. Select the picture <i>pictu_5_window_00.pdl</i> and the <i>target Object in Picture</i>, the object <i>Picture Window1</i> and the object property <i>Picture Name</i>.</p> <p>Apply the settings by clicking on the <i>OK</i> button.</p> 

Step	Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name
8	Use the  to select the configured <i>Button1</i> object and duplicate it via <i>Edit</i> → <i>Duplicate</i> . Repeat this procedure one more time. We now have two more buttons, <i>Button2</i> and <i>Button3</i> . At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , change the configured <i>direct connection</i> . For <i>Button2</i> , set the <i>source</i> to <i>pictu_5_window_01.pdl</i> and for <i>Button3</i> to <i>pictu_5_window_02.pdl</i> .

C-Action at Static Text1

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
char *name = lpszPictureName;
char *pdest;
int ch = '.';

//check if picture path contains char
pdest = strrchr( lpszPictureName, ch );
//read only picture name without path
if ( pdest == NULL ) return lpszPictureName;
else {
name = strcpy(name, strrchr(name, '.')+1);
return name;
}
}
```

- Declare the *C* variables.
- Check whether *lpszPictureName* only contains the picture name. This is done via the *strrchr* function. This function searches through *lpszPictureName*. If the picture is displayed in a *Picture Window*, *lpszPictureName* contains the picture name with the complete path of the picture.
- In the first case, return *lpszPictureName* directly as the return value.
- In the second case, read only the picture name from the picture path and return this name as the return value.

Note for the General Application

The following adaptations must be made before the general application:


- The *Static Text1* object can be transferred directly to any other *Picture Window*. This object is also suitable for being stored in the project library. In this way, it can simply be inserted into any picture by dragging and dropping it.
- For the direct connection at the *Button1* object, the picture name to be displayed and the object name of the *Picture Window* must be adapted.
- Pictures to be displayed, picture contents and *Picture Windows* must be laid out to meet your own requirements. The height and width of the picture and the *Picture Window* should agree.

3.2.2 Opening a Picture via the Dynamic Wizard (example 02)


Task Definition

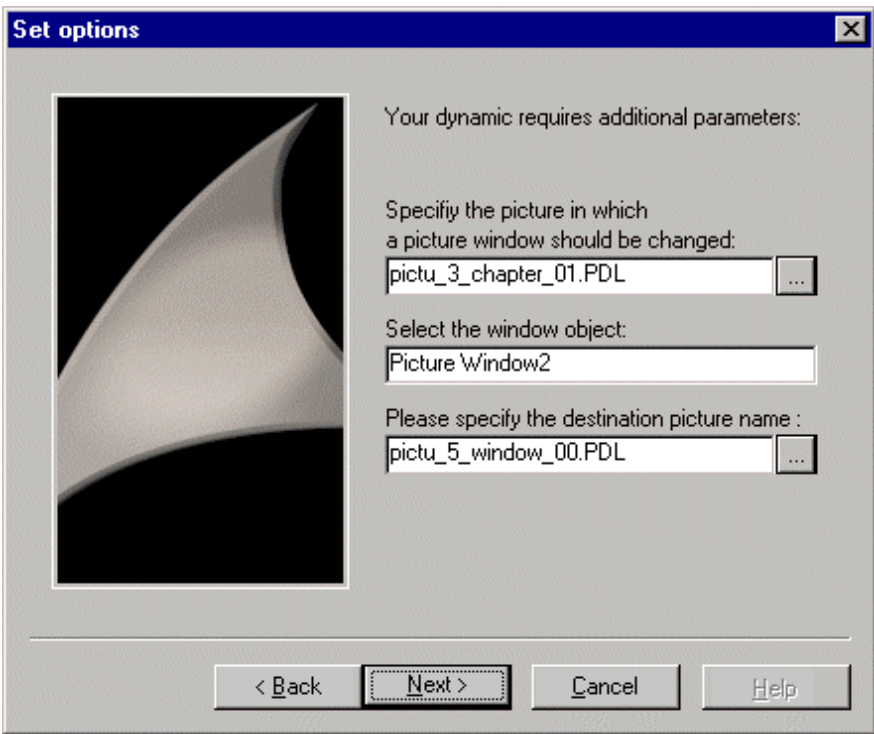
In the *Picture Window*, the picture change is to be performed via a mouse-operated *Button*. The configuration is to be performed with the *Dynamic Wizard*.


Implementation Concept

For the implementation, we will use a Windows Object → *Button*, which changes the picture displayed by the Smart Object → *Picture Window* when it is pressed with the . We will use the pictures already configured in the previous example.

Implementation in the WinCC Project

Step	Procedure: Opening a Picture via the Dynamic Wizard
1	In a picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, this is the <i>Picture Window2</i> . Adjust the dimension of the <i>Picture Window</i> to the screen size and set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . Via <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_01.pdl</i> picture.
2	If the <i>Dynamic Wizard</i> is not displayed, activate it from <i>View</i> → <i>Toolbars</i> .
3	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button4</i> object is used. While the object is highlighted, select the <i>Picture Functions</i> tab and then the <i>Picture Change in Window</i> entry via a  from the <i>Dynamic Wizard</i> . From the <i>Select Trigger</i> page of the <i>Dynamic Wizard</i> , select <i>Right Mouse Button</i> and go to the next page by clicking on the <i>Next</i> . Complete the <i>Set Options</i> page as follows:



Step	Procedure: Opening a Picture via the Dynamic Wizard
	Via the <i>Button</i>  , the pictures available to the project are accessed. Confirm the <i>Finished!</i> page by clicking on the <i>Finish</i> button.
4	Configure two additional <i>Windows Objects</i> → <i>Buttons</i> . In this sample, these are the <i>Button5</i> and <i>Button6</i> objects. Apply the <i>Dynamic Wizard</i> to these buttons as well. In the <i>Set Options</i> page for <i>Button5</i> , set <i>pictu_5_window_01.pdl</i> as the <i>Destination Picture Name</i> , and <i>pictu_5_window_02.pdl</i> for <i>Button6</i> .

C-Action generated by the Dynamic Wizard

```
#include "apdefap.h"
void OnRButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
static char szPicture[22] = "pictu_5_window_00.PDL";
static char* tmp = &szPicture[0];
PDLRTSetPropEx(PDLRT_AM_PICT&BS, "pictu_3_chapter_01",
"Picture Window2",
"PictureName", VT_LPSTR, &tmp, NULL, NULL, 0, NULL, NULL);
}
```

Note for the General Application

The following adaptations must be made before the general application:


- The *Dynamic Wizard* settings must be adapted to meet your own requirements.

3.2.3 Opening a Picture via an Internal Function (example 02)

Task Definition

In the *Picture Window*, the picture change is to be performed via a mouse-operated *Button*. The configuration at the *Button* is to be performed via a *C-Action*.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, which changes the picture displayed in the *Smart Object* → *Picture Window* when it is pressed with the . We will use the pictures from the previous example.

Implementation in the WinCC Project

Step	Procedure: Opening a Picture via an Internal Function
1	In a picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, this is the <i>Picture Window2</i> . Adjust the dimension of the <i>Picture Window</i> to the screen size and set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . Via <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_01.pdl</i> picture.
2	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button4</i> object is used. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create the <i>C-Action</i> for the picture change. Configure two additional <i>Buttons</i> . In this sample, these are the <i>Button5</i> and <i>Button6</i> objects, which are equipped with an appropriately modified <i>C-Action</i> .

C-Action at Button4

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetPictureName("pictu_3_chapter_01.PDL",
"Picture Window2",
"pictu_5_window_00");
}
```

- Via the *internal function SetPictureName*, switch the *pictu5_window_00.pdl* picture into the *Picture Window2* object. *pictu_3_chapter_01.pdl* is the name of the picture in which the *Picture Window* is located.

Note for the General Application

The following adaptations must be made before the general application:


- The parameters of the *internal function SetPictureName* must be adapted to meet your own requirements.

3.2.4 Single Picture Change via the Dynamic Wizard (example 03)



Task Definition

Via a mouse-operated *Button*, the picture displayed in runtime is to be changed. The configuration is to be implemented using the *Dynamic Wizard*.

Implementation Concept

For the implementation, a *Windows Object* → *Button* is used, which is clicked with the  to change the picture displayed in runtime.

Implementation in the WinCC Project

Step	Procedure: Single Picture Change via the Dynamic Wizard
1	<p>In this sample, the picture change is performed from the <i>pictu_0_startpicture_00.pdl</i> to the <i>pictu_3_chapter_01a.pdl</i> picture. In the sample project, the <i>pictu_0_startpicture_00.pdl</i> picture is always selected and from there, picture changes are only performed in windows. By using the <i>C-Action</i> generated by the <i>Dynamic-Wizard</i>, the entire picture system displayed in runtime is replaced by the one called. Changing back to <i>pictu_0_startpicture_00.pdl</i> is comparable to completely restarting the picture project.</p>
2	<p>In the picture, configure a <i>Windows Object</i> → <i>Button</i>. In this sample, the <i>Button7</i> object is used.</p> <div data-bbox="488 1098 1118 1171" style="text-align: center; background-color: #f4a460; padding: 5px;"> Picture Change via object name >> </div>
3	<p>While the object is highlighted, select the <i>Picture Functions</i> tab and then the <i>Single Picture Change</i> entry via a  from the <i>Dynamic Wizard</i>. From the <i>Select Trigger</i> page of the <i>Dynamic Wizard</i>, select the <i>Left Mouse Button list</i> entry and continue to the next page by clicking on the <i>Next</i> button. Complete the <i>Set Options</i> page as follows:</p> <div data-bbox="483 1360 935 1461" style="border: 1px solid #ccc; padding: 5px;"> <p>Please specify the new picture name:</p> <input data-bbox="495 1409 885 1444" type="text" value="pictu_3_chapter_03a.PDL"/> </div> <p>Via the <i>Button</i> , a list of all available pictures in the project will be displayed. Confirm the <i>Finished!</i> page by clicking on the <i>Finish</i> button.</p>
4	<p>If the picture change is performed in the sample project, click on the button</p> <div data-bbox="488 1587 1118 1661" style="text-align: center; background-color: #f4a460; padding: 5px;"> << Back </div> <p>to return to the project.</p>

C-Action generated by the Dynamic Wizard

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
OpenPicture("pictu_3_chapter_01a.PDL");
}
```

- The *Dynamic Wizard* generates a *C-Action*. This *C-Action* uses the *Standard Function OpenPicture* to switch the *pictu_3_chapter_01a.pdl* picture into runtime. The *C-Action* generated can also be programmed by the user.

Note for the General Application

The following adaptations must be made before the general application:

- The *Dynamic Wizard* settings must be adapted to meet your own requirements.

3.2.5 Single Picture Change via a Direct Connection (example 04)




This sample of the *Project_CreatePicture* project is accessed by clicking on the *Buttons* displayed above.


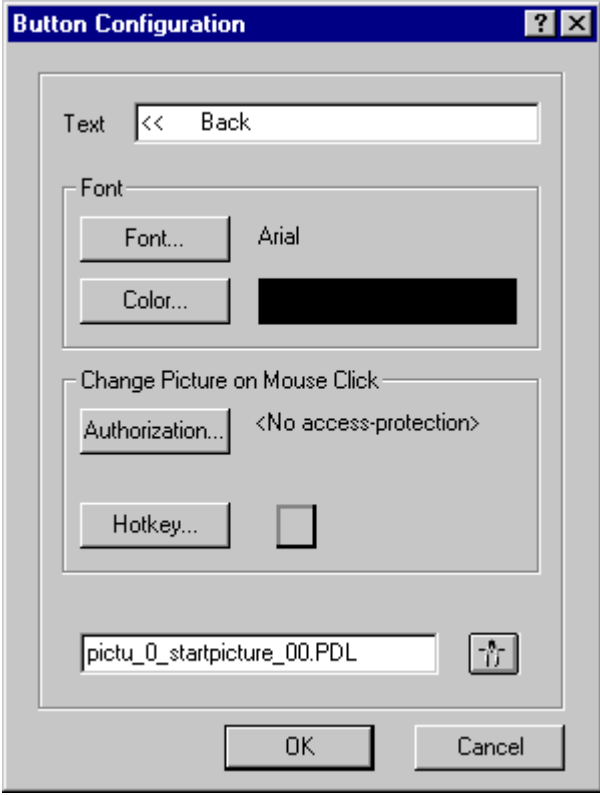
Task Definition

In contrast to the previous examples, clicking on a mouse-operated *Button* will change the entire picture. This will not just change the content of a picture window, but open a new picture.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, which will perform the picture change when clicked with the . The configuration is carried out via a *Direct Connection*.

Implementation in the WinCC Project

Step	Procedure: Single Picture Change via a Direct Connection
1	In this sample, a picture change from the <i>pictu_3_chapter_01a.pdl</i> to the <i>pictu_0_startpicture_00.pdl</i> picture is performed.
2	<p>In the picture, configure a <i>Windows Object</i> → <i>Button</i>. In this sample, the <i>Button7</i> object is used.</p> 
3	<p>In the <i>Change Picture on Mouse Click</i> section of the <i>button Configuration</i> dialog, select the <i>pictu_0_startpicture_00</i> picture using the selection button. This automatically generates a <i>direct connection</i> at <i>Event</i> → <i>Mouse</i> → <i>Mouse Action</i>. This connection can also be generated from the <i>Object Properties</i> dialog.</p> 

Note for the General Application

The following adaptations must be made before the general application:

- In the *direct connection* at the *Button7* object, the picture name and the object name of the picture window must be adapted.

3.2.6 Opening a Picture via the Object Name and an Internal Function (05)




This sample of the *Project_CreatePicture* project is accessed by clicking on the *Buttons* displayed above.

Task Definition

In the *Picture Window*, the picture change is to be performed via a mouse-operated button. The *Button* is to recognize which picture it to be called via its object name. The button can therefore only be reused after being copied if its object name is changed.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, which changes the picture displayed in the *Smart Object* → *Picture Window* when it is pressed with the . We will use the pictures already configured in the previous example. The names of these pictures comprise two components: a text section and a picture number.

Implementation in the WinCC Project

Step	Procedure: Opening a Picture via the Object Name and an Internal Function
1	In a picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, the <i>Picture Window1</i> object is used. Adjust the dimension of the <i>Picture Window</i> to match the size of the previously created pictures. To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . Via <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_00.pdl</i> picture.
2	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button0</i> object is used. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a <i>C-Action</i> that reads the name and number of the <i>Button</i> and displays the desired picture according to the set name conventions.
3	Duplicate the <i>Button0</i> object two time and change the object name of <i>Button1</i> and <i>Button2</i> .

C-Action at Button0

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
char   name[30];
int    number;
int    ch = 'n';
char   *pdest;

//check if object name contains character
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
else {
//read object number
number = atoi(strrchr(lpszObjectName, 'n')+1);
//generate picture name
sprintf(name, "pictu_5_window_%02d.PDL", number);
//set picture name
SetPictureName("pictu_3_chapter_01a.PDL", "Picture Window1", name);
}
}
```

- Declare the *C* variables.
- Check whether the object has been named in accordance with the agreed guidelines. The objects are given the name [button]+[number of the picture to be called].
- Output of an error message, if the character in front of the number, namely n, is not found.
- Reading the number of the button name. The *strrchr* function searches backward through the name for the character n. The character string following *n* is taken and converted to an integer value by the *atoi* function.
- The *sprintf* function uses the picture name and picture number components to generate the complete picture name to be called by the *Button*.
- Via the *internal function SetPictureName*, the picture to be called is switched into the *Picture Window1* object.

Note for the General Application

The following adaptations must be made before the general application:

- The *C-Action* at the *Button* and the assignment of the object name must be adapted to meet your own name conventions. Make sure that these name conventions are always strictly observed, both in object names and picture names, to guarantee trouble-free access to the desired picture.

3.2.7 Opening a Picture via the Object Name and a Tag Connection with Display of the Picture Name (example 06)




This sample of the *Project_CreatePicture* project is accessed by clicking on the *Buttons* displayed above.

Task Definition

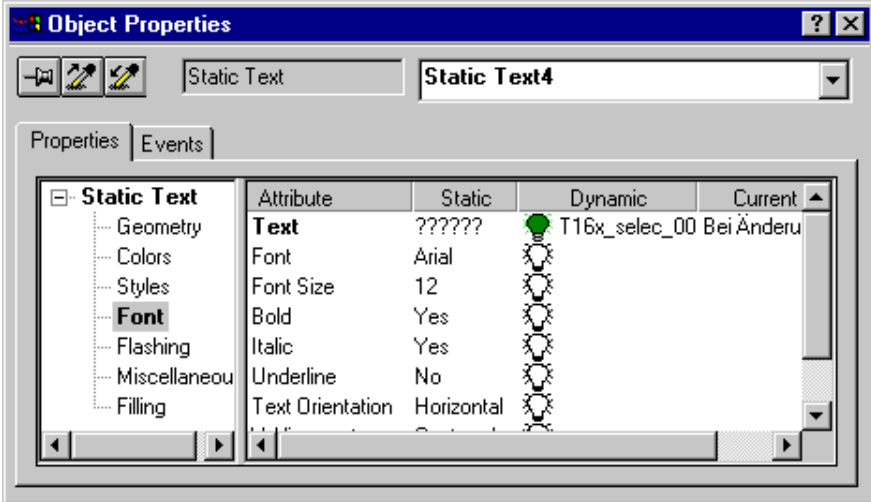
In the *Picture Window*, the picture change is to be performed via a mouse-operated button. The *Button* is to recognize which picture it to be called via its object name. The button can therefore only be reused after being copied if its object name is changed. The picture name is to be stored in a text tag and displayed in a text field which is not in the actual picture.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, which changes the picture displayed in the *Smart Object* → *Picture Window* when it is pressed with the . We will use the pictures already configured in the previous example. The names of these pictures comprise two components: a text section and a picture number. In addition, a *Standard Object* → *Static Text* for the display of the picture name is used.

Implementation in the WinCC Project

Step	Procedure: Opening a Picture via the Object Name and a Tag Connection with Picture Name Display
1	In Tag Management, create a tag of the <i>Text Tag 16-Bit Character Set</i> type. In this sample, the <i>T16x_selec_00</i> tag is used. This tag contains the name of the picture displayed in the picture window.
2	Open the properties dialog of the <i>pic_chapter_01a.pdl</i> picture object. At <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> , configure a C-Action that assigns the picture name <i>pictu_5_window_01.pdl</i> to the <i>T16x_selec_00</i> tag. This corresponds to the picture that is to be displayed the first time a picture is opened.
3	In a picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, the <i>Picture Window2</i> is used. Adjust the dimension of the <i>Picture Window</i> to match the size of the previously created pictures. To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_01.pdl</i> and create a <i>tag connection</i> to the <i>T16x_selec_00</i> tag.
4	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button_0</i> object is used. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a C-Action that reads the name and number of the <i>Button</i> and assigns the name to the internal tag <i>T16x_selec_00</i> .
5	Duplicate the <i>Button_0</i> object two times and change the object name of <i>Button_1</i> and <i>Button_2</i> .

Step	Procedure: Opening a Picture via the Object Name and a Tag Connection with Picture Name Display																																
6	<p>In the picture, configure a <i>Smart Object</i> → <i>Static Text</i> above the <i>Picture Window Picture Window2</i>. In this sample, the <i>Static Text4</i> object is used. Set <i>Properties</i> → <i>Font</i> → <i>Bold</i> to <i>Yes</i>. At <i>Properties</i> → <i>Font</i> → <i>Text</i>, delete the entered text from the <i>Static</i> column and create a <i>tag connection</i> to the <i>T16x_selec_00</i> tag. Set the update to <i>Upon Change</i>. Deleting the static entry prevents an incorrect text from being output at the moment of picture buildup.</p>  <table border="1" data-bbox="560 693 1364 976"> <thead> <tr> <th>Attribute</th> <th>Static</th> <th>Dynamic</th> <th>Current</th> </tr> </thead> <tbody> <tr> <td>Text</td> <td>??????</td> <td></td> <td>T16x_selec_00 Bei Änderung</td> </tr> <tr> <td>Font</td> <td>Arial</td> <td></td> <td></td> </tr> <tr> <td>Font Size</td> <td>12</td> <td></td> <td></td> </tr> <tr> <td>Bold</td> <td>Yes</td> <td></td> <td></td> </tr> <tr> <td>Italic</td> <td>Yes</td> <td></td> <td></td> </tr> <tr> <td>Underline</td> <td>No</td> <td></td> <td></td> </tr> <tr> <td>Text Orientation</td> <td>Horizontal</td> <td></td> <td></td> </tr> </tbody> </table>	Attribute	Static	Dynamic	Current	Text	??????		T16x_selec_00 Bei Änderung	Font	Arial			Font Size	12			Bold	Yes			Italic	Yes			Underline	No			Text Orientation	Horizontal		
Attribute	Static	Dynamic	Current																														
Text	??????		T16x_selec_00 Bei Änderung																														
Font	Arial																																
Font Size	12																																
Bold	Yes																																
Italic	Yes																																
Underline	No																																
Text Orientation	Horizontal																																

C-Action at Open Picture

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetTagChar("T16x_selec_00", "pic_window_01.pll");
}
```

- Assignment of the picture name via the die *internal function SetTagChar*.

C-Action at Button_0

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
char name[30];
int number;
int ch = '_';
char *pdest;

//check if object name contains character
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameErrcr"));
else {
//read object number
number = atoi(strrchr(lpszObjectName, '_')+1);
//generate picture name
sprintf(name, "pictu_5_window_%02d.PDL", number);
//set tag which contains picture name
SetTagChar( "T16x_selec_00", name);
}
}
```

- Declare the *internal tags*.
- Check whether the object has been named in accordance with the agreed guidelines. The objects are given the name [button]+[_]+[number of the picture to be called].
- Output of an error message if the character _ is not found.
- Reading the number of the button name. The function *strrchr* searches backward through the name for the character _. The character string following the _ is taken and converted to an integer value by the *atoi* function.
- The *sprintf* function uses the picture name and picture number components to generate the complete picture name to be called by the *Button*.
- Via the *internal function SetTagChar*, the picture name to be called is transferred to the *T16x_selec_00* tag.


Note for the General Application

The following adaptations must be made before the general application:

- The *C-Action* at the *Button* and the assignment of the object name must be adapted to meet your own name conventions. Make sure that these name conventions are always strictly observed, both in object names and picture names, to guarantee trouble-free access to the desired picture.

3.3 Displaying a Picture Window

Zoom


The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_03.pdl* picture.

3.3.1 Hiding (Deselection) and Displaying (Selection) from outside the Picture Window (example 01)

Task Definition

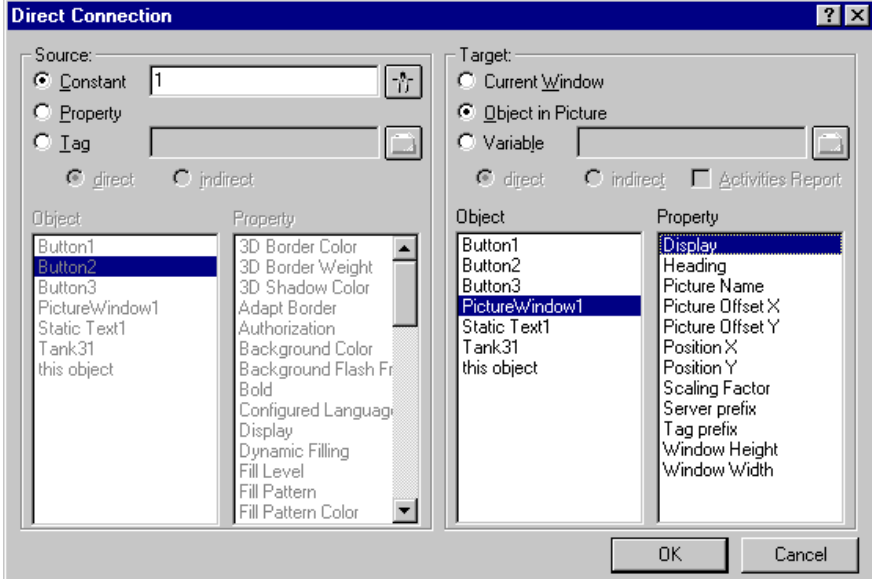
A picture window is to be displayed and hidden again via two mouse-operated *Buttons*.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons* that will display and hide the picture in the *Smart Object* → *Picture Window* when pressed with the .

Implementation in the WinCC Project

Step	Procedure: Hiding and Displaying from outside the Picture Window
1	Configure a picture that is to be displayed and hidden, e.g. a help text or an information box. In the sample, the <i>pictu_5_window_07</i> is used, a pure information box without additional control elements.
2	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> with the same geometric dimension as the previously created picture. In this sample, the <i>Picture Window1</i> object is used. Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 246 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 129. To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . To allow the window to be moved around, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i> . To hide the window in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_07.pdl</i> picture.

Step	Procedure: Hiding and Displaying from outside the Picture Window
3	<p>In the same picture, configure two additional <i>Windows Objects</i> → <i>Buttons</i>. In this sample, these are the <i>Button1</i> and <i>Button2</i> objects. For <i>Button1</i>, configure a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>. Connect the <i>source Constant</i> → <i>1</i> with the <i>target Object in Picture</i> → <i>Picture Window1</i> → <i>Display</i>. Apply the settings by clicking on the <i>OK</i> button.</p> 
4	<p>In the same manner as for <i>Button1</i>, configure a <i>direct connection</i> for <i>Button2</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>. As <i>Constant</i>, specify the value <i>0</i>.</p>

Note for the General Application

The following adaptations must be made before the general application:


- For the *direct connections* at the *Button1* and *Button2* objects, the picture name to be displayed and the object name of the *Picture Window* must be adapted.
- The supplied *pictu_5_window_07* picture can be transferred directly to another project after modifying its title and information text.

3.3.2 Displaying (Selection) from outside and Hiding (Deselection) from within the Picture Window (example 02)

Task Definition

A picture window is to be made visible by clicking on a mouse-operated *Button*. The picture window is to be hidden by clicking on a button within the *Picture Window*.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons* that will display and hide the picture in the *Smart Object* → *Picture Window* when pressed with the .

Implementation in the WinCC Project

Step	Procedure: Displaying (Selection) from outside and Hiding (Deselection) from within the Picture Window
1	Configure a picture that is to be displayed and hidden, e.g. a help text or an information box. In this sample, the <i>pictu_5_window_08</i> picture is used, an information box with an additional <i>Windows Object</i> → <i>Button</i> that deselects the picture. In this sample, the <i>Button1</i> object is used.
2	For <i>Button1</i> , configure a <i>direct connection at Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the source <i>Constant</i> → <i>0</i> with the target <i>Current Window</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
3	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> with the same geometric dimension as the previously created picture. In this sample, the <i>Picture Window2</i> object is used. Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to <i>246</i> and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to <i>129</i> . To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . To allow the window to be moved around, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i> . To hide the window in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> . Via <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_08.pdl</i> picture.
4	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, this is the <i>Button3</i> object. For <i>Button3</i> , create a <i>Direct Connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>1</i> with the <i>target Object in Picture</i> → <i>Picture Window2</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.

Note for the General Application

The following adaptations must be made before the general application:


- For the *direct connection* at the *Button3* object, the picture name to be displayed and the object name of the *Picture Window* must be adapted.
- The supplied *pictu_5_window_08* picture can be transferred directly to another project after modifying its title and information text. There is no need to make any changes to the *direct connection* at *Button1*.

3.3.3 Time-Controlled Hiding of a Picture (example 03)


Task Definition

A *Picture Window* is to be displayed and hidden using a mouse-operated *Button*. After a set time, the picture window is to be hidden automatically .

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button* that will display and hide the picture in the *Smart Object* → *Picture Window* when pressed with the .

Implementation in the WinCC Project

Step	Procedure: Time-Controlled Hiding of a Picture
1	<p>Configure a picture that is to be displayed and hidden, e.g. a help text or an information box. In this sample, the <i>pictu_5_window_09</i> picture is used, an information box without any additional control elements. To implement the time-controlled hiding of the <i>Graphic Object1</i>, a <i>C-Action</i> is configured at <i>Properties</i> → <i>Geometry</i> → <i>Position X</i>. This <i>C-Action</i> can be positioned anywhere, since only one <i>Trigger</i> is required. Set <i>1 s</i> as the trigger.</p> 
2	<p>In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> with the same geometric dimension as the previously created picture. In this sample, the <i>Picture Window3</i> object is used. Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to <i>246</i> and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to <i>129</i> . To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i>. To allow the window to be moved around, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i>. To hide the window in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i>. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the <i>pictu_5_window_09.pdl</i> picture.</p>
3	<p>Configure a button, in this sample, the <i>Button4</i> object is used. Configure a <i>C-Action</i> for <i>Button4</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that displays hides the <i>Picture Window</i>.</p>

C-Action at Graphic Object1

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
static int i = 0;

//count time
i++;
//if maximum time is reached
if (i>5) SetVisible("pictu_3_chapter_03.PDI", "Picture Window3", 0);
return 0;
}
```

- Declare the static *C variable*. This tag retains its value during the time the picture is open.
- Increment the static *C variable* each time the program is called.
- If *i* exceeds the value of 5, i.e. for a trigger set to 1 s after 5 s, the *Picture Window* will be hidden.
- The return value is the X position of the *Graphic Object1*, since the *C-Action* is positioned at this property, but the property itself is not to be changed.

C-Action at Button4

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
//set visibility in complement state
SetVisible(lpszPictureName, "Picture Window3",
!(SHORT)!GetVisible(lpszPictureName, "Picture Window3"));
}
```

- The visibility of the *Picture Window Picture Window3* is assigned the opposite status of the current visibility by the *internal function SetVisible*. The current status is queried by the *internal function GetVisible*.



Note for the General Application

The following adaptations must be made before the general application:


- For the *C-Action* at the *Button4* object, the picture name to be displayed and the object name of the *Picture Window* must be adapted.
- The supplied *pictu_5_window_09* picture can be transferred directly to another project after modifying its title and information text. At the *C-Action* of the *Graphic Object1*, the time until the picture is hidden can be user-defined by changing the trigger or by changing the condition in the *if* statement.

3.3.4 Displaying a Picture Window while the Right Mouse Button is Pressed (example 04)

Task Definition

A *Picture Window* is to be displayed while a *Button* is pressed via a  and be hidden again if the  is released.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, with which the picture in the *Smart Object* → *Picture Window* is made visible while is it being pressed with the .

Implementation in the WinCC Project

Step	Procedure: Displaying a Picture Window while the Right Mouse Button is Pressed
1	Configure a picture that is to be displayed and hidden, e.g. a help text or an information box. In the sample, the <i>pictu_5_window_07</i> picture is used, a pure information box without any additional control elements.
2	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> with the same geometric dimension as the previously created picture. In this sample, the <i>Picture Window1</i> object is used. Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 246 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 129. To have the window displayed with a border in runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> . To allow the window to be moved around, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , set the <i>pictu_5_window_07.pdl</i> picture.
3	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> ; in this sample, this is the <i>Button5</i> object. For <i>Button5</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Right</i> . Connect the <i>source Constant</i> → 1 with the <i>target Object in Picture</i> → <i>Picture Window4</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
4	In the same manner, create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Release Right</i> . As <i>Constant</i> , specify the value 0.


Note for the General Application

The following adaptations must be made before the general application:

- For the *direct connections* at *Button5*, the picture name to be displayed and the object name of the *Picture Window* must be adapted.
- The supplied *pictu_5_window_07* picture can be transferred directly to another project after modifying its title and information text.

3.3.5 Configuring Information Boxes with the Wizard (example 05)



This sample is accessed by selecting the Button displayed above with the . The sample is configured in the *pictu_3_chapter_03a.pdl* picture.


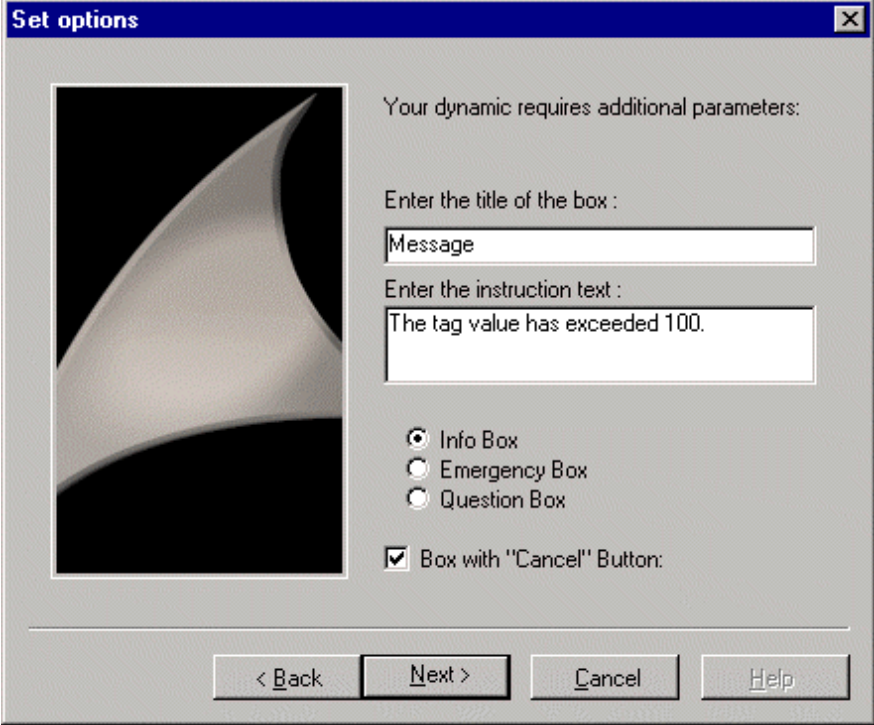
Task Definition

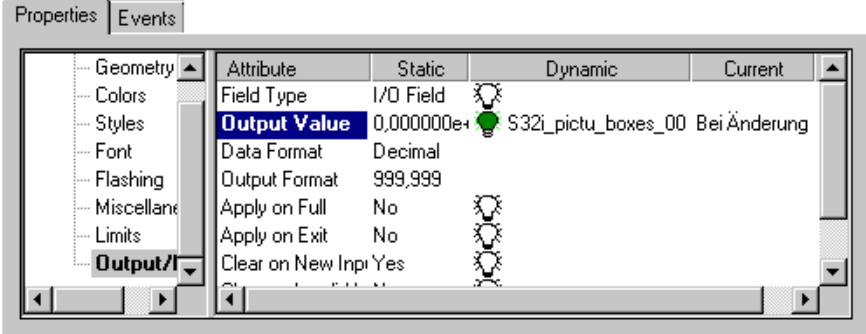
An information (instruction) box is to be displayed if a tag exceeds the value of 100 and an emergency box if this tag exceeds the value of 150.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Slider Object* to enter the tag value and a *Smart Object* → *I/O Field* to display the tag value.

Implementation in the WinCC Project

Step	Procedure: Configuring Information Boxes with the Wizard
1	If the <i>Dynamic Wizard</i> is not displayed, activate it from <i>View</i> → <i>Toolbars</i> .
2	<p>In a picture, configure a <i>Smart Object</i> → <i>I/O Field</i>. In this sample, the <i>I/O Field1</i> object is used. While the object is highlighted, select the <i>Picture Functions</i> tab and then the <i>Display Instruction Box</i> entry via a  from the <i>Dynamic Wizard</i>. From the <i>Select Trigger</i> page of the <i>Dynamic Wizard</i>, select the <i>Left Mouse Button list</i> entry and continue to the next page by clicking on the <i>Next</i> button. Complete the <i>Set Options</i> page as follows:</p>  <p>Confirm the <i>Finished!</i> page by clicking on the <i>Finish</i> button.</p>
3	Use the <i>Dynamic Wizard</i> again for the <i>I/O Field1</i> . On the <i>Select Trigger</i> page, select <i>Right Mouse Button</i> ; on the <i>Set Options</i> page, select the <i>Emergency Box</i> radio-button and enter the text for the display.
4	In <i>Tag Management</i> , create a tag of the <i>Signed 32-Bit Value</i> type. In this sample, the <i>S32i_pictu_boxes_00</i> tag is used.
5	In the same picture, configure a <i>Windows Object</i> → <i>Slider Object</i> . In this sample, this is the <i>Slider Object1</i> . For the <i>Slider Object1</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> . Connect the <i>source Property</i> → <i>Slider Object1</i> → <i>Process Driver Connection</i> with the <i>Target Variable</i> → <i>S32i_pictu_boxes_00</i> . Apply the settings by clicking on the <i>OK</i> button.

Step	Procedure: Configuring Information Boxes with the Wizard
6	<p>For the <i>I/O Field1</i> object, create a <i>Tag Connection</i> at <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i> to the tag <i>S32i_pictu_boxes_00</i> and trigger upon change.</p> 
7	<p>For the <i>I/O Field1</i> object, create a <i>C-Action</i> at <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Output Value</i> that displays an information box if the <i>S32i_pictu_boxes_00</i> tag exceeds the value 100 and an emergency box if the value 150 is exceeded. The <i>C-Actions</i> generated by the <i>Dynamic Wizard</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> and <i>Press Right</i> can be copied and pasted into this <i>C-Action</i>.</p>
8	<p>Delete the <i>C-Actions</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> and <i>Press Right</i>.</p>

C-Action at I/O Field1

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l
{
int a;
static int i = 0, j = 0;

//get tag value
a=GetTagDWord("S32i_pictu_boxes_00");

//set visible info box
if ((a>100)&&(i==0)) {
    i=1;
    MessageBox(NULL, "Der Variablenwert hat\r\n100 überschritten",
        "Hinweis", MB_OK|MB_ICONEXCLAMATION|MB_SETFOREGROUND);
} //if
if (a<=100) (i=0);

//set visible emergency box
if ((a>150)&&(j==0)) {
    j=1;
    MessageBox(NULL, "Der Variablenwert hat\r\n150 überschritten",
        "Achtung!!!", MB_OK|MB_ICONSTOP|MB_SETFOREGROUND);
} //if
if (a<=150) (j=0);
}
```

- Read in the tag value using the *internal function* *GetTagDWord*.
- If 100 is exceeded, the information box is displayed using the *C-Action* generated by the *Dynamic Wizard*. If 100 is exceeded, the information box will only be closed again, if 100 is fallen below, i.e. the static *C variable* *i* has been reset to zero.
- If 150 is exceeded, the emergency box is displayed using the *C-Action* generated by the *Dynamic Wizard*. If 150 is exceeded, the emergency box will only be closed again, if 150 is fallen below, i.e. the static *C variable* *j* has been reset to zero.


Note for the General Application

The following adaptations must be made before the general application:


- For the *C-Action* at the *I/O Field1* object, the variable name must be adapted.
- The text displayed in the information and emergency boxes must be adapted to meet your own requirements.

3.3.6 Displaying a Dialog for Text Input (example 06)



This sample is accessed by selecting the *Button* displayed above with the . The sample is configured in the *pictu_3_chapter_03a.pdl* picture.

Task Definition

When pressing a *Button* with the , the text input dialog is to be displayed. The text entered is displayed in the picture.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button* to open the dialog and a *Standard Object* → *Static Text* to display the text. For the text input in the dialog, we will use a *Smart Object* → *I/O Field* and two *Windows Objects* → *Buttons* to apply or cancel the input.

Implementation in the WinCC Project

Step	Procedure: Displaying a Dialog for Text Input
1	In Tag Management, create two tags of the <i>Text Tag 16-Bit Character Set</i> type. In this sample, the <i>T16i_pictu_win_00</i> and <i>T16i_pictu_win_01</i> tags are used.
2	Configure a picture in which text input is to be carried out. In the sample, the <i>pictu_5_window_17.pdl</i> picture is used.
3	In this picture, configure a <i>Smart Object</i> → <i>I/O Field</i> . In its <i>configuration dialog</i> , select the <i>T16i_pictu_win_01</i> tag and set the trigger to <i>Upon Change</i> . Set the <i>Property</i> → <i>Output/Input</i> → <i>Data Format</i> to <i>String</i> and the <i>Property</i> → <i>Output/Input</i> → <i>Apply on Exit</i> to <i>Yes</i> . This means that the ENTER key does not have to be pressed to accept the text entered.
4	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button1</i> is used. This button is used to apply the text entered. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a <i>direct connection</i> with the <i>source Variable</i> <i>T16i_pictu_win_01</i> and the <i>target Variable</i> <i>T16i_pictu_win_00</i> . At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> , configure a <i>direct connection</i> that hides the picture.
5	Configure another <i>Windows Object</i> → <i>Button</i> ; in this sample, this is the <i>Button2</i> object. This button is used to cancel the input, the text entered previously is retained. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a <i>direct connection</i> with the <i>source Variable</i> <i>T16i_pictu_win_00</i> and the <i>target Variable</i> <i>T16i_pictu_win_01</i> . This direct connection transfers the content of <i>T16i_pictu_win_00</i> (containing the previous text) to <i>T16i_pictu_win_01</i> . At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> , configure a <i>direct connection</i> that hides the picture.

Step	Procedure: Displaying a Dialog for Text Input
6	In a second picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, the <i>Picture Window1</i> object is used. Adjust the dimension of the <i>Picture Window</i> to match the size of the picture just created. If the <i>Picture Window</i> is to be displayed with a border, the <i>Height</i> and <i>Width</i> of the <i>Picture Window</i> must be 10 pixels greater than those of the picture. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , enter <i>pictu_5_window_17.pdl</i> .
7	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In the sample, this is the <i>Button1</i> object. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create a direct connection. Connect the <i>source Constant</i> → <i>1</i> with the target <i>Object in Picture</i> → <i>Picture Window1</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
8	In the same picture, configure a <i>Standard Object</i> → <i>Static Text</i> . In the sample, this is the <i>Static Text1</i> object. At <i>Properties</i> → <i>Font</i> → <i>Text</i> , configure a <i>tag connection</i> to the <i>T16i_pictu_win_00</i> tag and trigger it upon change.


Note for the General Application

The following adaptations must be made before the general application:

- The *pictu_5_window_17.pdl* picture can be used directly for the text input, the *C-Actions* at the *Buttons*, however, must be adapted to match your own variable names.

3.4 Operator-Control Enable

Operator Panels


The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_02.pdl* picture.

3.4.1 Exiting Runtime and System (example 01)





Task Definition

Two mouse-operated *Buttons* are used to select two control windows, which are used to either exit runtime only or the entire system.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons*, which each display a picture in a *Smart Object* → *Picture Window* when pressed with the . In the individual pictures, two *Windows Objects* → *Buttons* make it possible to either call the corresponding system function or to cancel the procedure.

Implementation in the WinCC Project

Step	Procedure: Exiting Runtime and System
1	Configure a picture, which is going to be used to exit runtime. In the sample, the <i>pictu_5_window_04.pdl</i> picture is used.
2	In this picture, configure a <i>Windows Object</i> → <i>Button</i> ; in the sample, the <i>Button1</i> object is used. While the object is highlighted, select the <i>System Functions</i> tab and then the <i>Exit WinCC or Windows</i> entry via a  from the <i>Dynamic Wizard</i> . From the <i>Select Trigger</i> page of the <i>Dynamic Wizard</i> , select the <i>Left Mouse Button</i> entry and continue to the next page by clicking on the  <i>Next button</i> . On the <i>Set Options</i> page, select <i>Exit Windows</i> . Confirm the <i>Finished!</i> page by clicking on the <i>Finish</i> button.
3	Configure another <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button2</i> object is used. This button is used to cancel the procedure. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a <i>direct connection</i> that will hide the picture.
4	Configure another picture, which is going to be used to shut down the system. In the sample, the <i>pictu_5_window_03.pdl</i> picture is used.
5	In this picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button1</i> is used. While the object is highlighted, select the <i>System Functions</i> tab and then the <i>Exit WinCC Runtime</i> entry via a  from the <i>Dynamic Wizard</i> . From the <i>Select Trigger</i> page of the <i>Dynamic Wizard</i> , select the <i>Left Mouse Button</i> entry and continue to the next page by clicking on the  <i>Next button</i> . Confirm the <i>Finished!</i> page by clicking on the <i>Finish</i> button.
6	Configure another <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button2</i> object is used. This button is used to cancel the procedure. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a <i>direct connection</i> that will hide the picture.

Step	Procedure: Exiting Runtime and System
7	<p>In another picture, configure two <i>Smart Objects</i> → <i>Picture Windows</i>; in this picture, the <i>Picture Window1</i> and <i>Picture Window2</i> object are used that are arranged one on top of the other. Adjust the dimension of the <i>Picture Windows</i> to match the size of the pictures just created. If the <i>Picture Windows</i> are to be displayed with borders, the <i>Height</i> and the <i>Width</i> of the <i>Picture Windows</i> must be set 10 pixels greater than those of the pictures, in order to display the entire picture. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, enter the respective picture names. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display to No</i>.</p>
8	<p>In the same picture, configure two <i>Windows Objects</i> → <i>Buttons</i>. In this sample, these are the <i>Button1</i> and <i>Button2</i> objects. For <i>Button1</i>, create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>. Connect the <i>source Constant</i> → <i>1</i> with the <i>target Object in Picture</i> → <i>Picture Window1</i> → <i>Display</i>. Apply the settings by clicking on the <i>OK</i> button. In the same manner, create a <i>direct connection</i> for <i>Button2</i>, but set as <i>target Object in Picture</i> → <i>Picture Window2</i> → <i>Display</i>.</p>

Note for the General Application

The following adaptations must be made before the general application:


- The pictures for exiting the system and runtime can be applied directly to other projects.
- At the *Buttons* for calling the *Picture Windows*, the object names of the *Picture Windows* in the *direct connections* must be adapted.

3.4.2 Operator-Control Enable, Logon with Default Box (example 02)





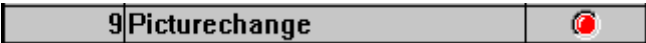
Task Definition


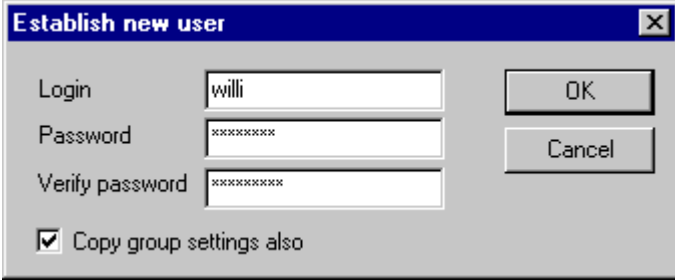



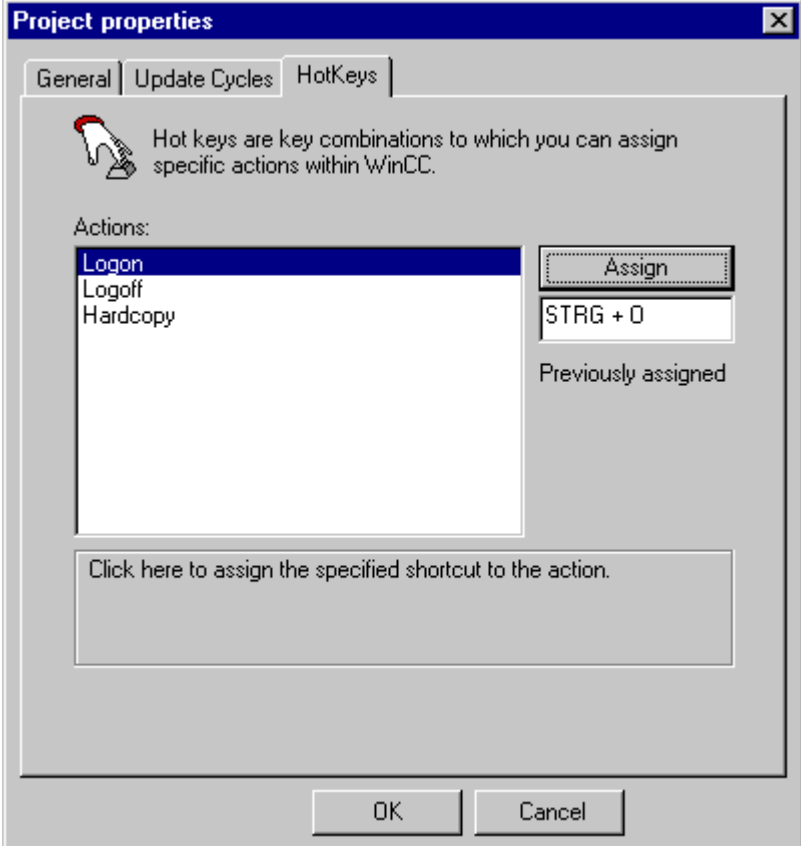
Via two *Buttons*, a picture change is only to be performed if the user has the appropriate authorization.


Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons*, which each display a different picture in a *Smart Object* → *Picture Window* when pressed with the . The settings required for assigning user authorizations are made in the *User Administrator* editor.

Implementation in the WinCC Project

Step	Procedure: Operator-Control Enable, Logon with Default Box
1	<p>In the <i>WinCC Explorer</i>, open the <i>User Administrator</i> editor by  on it and then selecting <i>Open</i> from the pop-up menu.</p> 
2	<p>Via the button, create a new user group and assign a name to it; in this sample, we will use the name <i>service</i>.</p>
3	<p>Via the <i>Table</i> → <i>Add new Authorization Level</i> menus, define the authorization level <i>Picture Change</i> as line number 9. This authorization level is assigned to the <i>service</i> group. To do so, select the group with the . In the table containing the line <i>Picture Change</i>, select the radio-button in the <i>Authorization</i> column with a .</p> <p>An authorization level assigned to a group or user is marked by a red dot next to it in the <i>Authorization</i> column.</p> 

Step	Procedure: Operator-Control Enable, Logon with Default Box
4	<p>Via the <i>Button</i> , create a new user for the <i>service</i> user group. In the sample project, a user named <i>willi</i> with the password <i>Project_CreatePicture</i> has been created. Activate the <i>Copy Group Settings Also</i> check-box to transfer the authorization levels valid for the group to the user.</p>  <p>Via the <i>File</i>  <i>Exit</i> menu, close the <i>User Administrator</i> editor.</p>
5	<p>In the WinCC Explorer,  on the project name to access the project properties dialog.</p> <p>In the window displayed, select the <i>HotKeys</i> tab and make the desired settings for calling the <i>Logon</i> and <i>Logoff</i> dialogs. To assign the hotkeys, use the  and click on the <i>Assign</i> button. In this sample, the key combination <i>CTRL+O</i> for <i>Logon</i> and <i>CTRL+F</i> for <i>Logoff</i> is used.</p> 

Step	Procedure: Operator-Control Enable, Logon with Default Box
6	In a picture, configure two <i>Windows Objects</i> → <i>Buttons</i> . In this sample, the <i>Button3</i> and <i>Button4</i> objects are used. Configure a <i>Smart Object</i> → <i>Picture Window</i> , into which pictures are inserted via <i>direct connections</i> at the two <i>Buttons</i> .
7	For the <i>Button3</i> and <i>Button4</i> objects, select the <i>Picture Change</i> user level at <i>Properties</i> → <i>Miscellaneous</i> → <i>User Level</i> and set the <i>Property</i> → <i>Miscellaneous</i> → <i>Operator-Control Enable</i> to <i>No</i> .
8	While the <i>Button3</i> object is highlighted, select the <i>Standard Dynamics</i> tab and then the <i>Operational if Authorized</i> entry via a  from the Dynamic Wizard. Complete the <i>Dynamic Wizard</i> by clicking on the <i>Finish</i> button. Repeat the same procedure for <i>Button4</i> .
9	In Tag Management, create the <i>@CurrentUser</i> system tag of the <i>Text Tag 16-Bit Character Set</i> type. The user name currently logged in is automatically assigned to this tag.
10	Trigger the <i>C-Actions</i> at <i>Button3</i> and <i>Button4</i> generated by the <i>Dynamic Wizard</i> upon change of this tag. This means that the <i>C-Action</i> will no longer be executed every 2 seconds, but only if the user name changes.

C-Action generated by the Dynamic Wizard

```
#include "apdefap.h"
BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN
{
#pragma code ("UseAdmin.DLL")
#include "pwrt_api.h"
#pragma code ()
#define NO_MESSAGEBOX 1
CMN_ERROR err;
DWORD pwlevel = 0;
pwlevel = (DWORD) GetPasswordLevel(lpszPictureName, lpszObjectName);
if (pwlevel==0)
return (TRUE);
else
return (PWRTCheckPermissionOnPicture(pwlevel, lpszPictureName, NO_MESSAGEBOX, &
}
```

Note for the General Application


The following adaptations must be made before the general application:
The names of the user groups and users, the logons and the passwords must be adapted.

3.4.3 Operator-Control Enable, Logon via a separate Dialog (example 03)

Task Definition

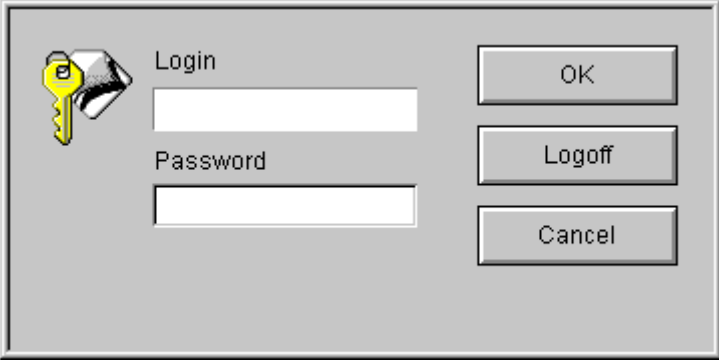
Via a Button, exiting runtime should only be possible if the user has the appropriate authorization. Via a Button, a dialog for logging in is to be displayed.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons*. With the first button, a *Smart Object* → *Picture Window* for the Logon is to be displayed when pressed with the . The second button is used to exit down runtime.

Implementation in the WinCC Project

Step	Procedure: Operator-Control Enable, Logon via a separate Dialog
1	In the <i>User Administrator</i> editor, create a new user group and give it a name; in this sample, the name user is used. At line number 10, define a new authorization level named Exit Runtime. This authorization level is assigned to the just created user group. Create a user for the group. In the sample project, a user named <i>ulrich</i> with the password <i>Project_CreatePicture</i> has been created.
2	In a picture, configure two <i>Windows Objects</i> → <i>Buttons</i> . In this sample, the <i>Button5</i> and <i>Button6</i> objects are used.
3	For <i>Button5</i> , configure the call of a <i>Smart Object</i> → <i>Picture Window</i> for ending runtime. In this sample, the <i>Picture Window Picture Window5</i> object is used.
4	For the <i>Button5</i> object, select the Exiting Runtime user level at Properties → Miscellaneous → User Level and set the Property → Miscellaneous → Operator-Control Enable to No.
5	Apply the <i>Operational if Authorized</i> Dynamic Wizard to <i>Button5</i> . Set the C-Action generated to be triggered by the <i>@CurrentUser</i> system tag.

Step	Procedure: Operator-Control Enable, Logon via a separate Dialog
6	<p>Configure a <i>Smart Object</i> → <i>Picture Window</i>. In this sample, the <i>Picture Window4</i> object is used. Set the <i>Property</i> → <i>Geometry</i> → <i>Window Height</i> to 360 and the <i>Property</i> → <i>Geometry</i> → <i>Window Height</i> to 180. Set the <i>Properties</i> → <i>Miscellaneous Moveable, Border, Title and Foreground</i> to <i>Yes</i>. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, select the <i>pictu_5_window_18.pdl</i> picture. This picture can be taken directly from the <i>Project_CreatePicture</i> project.</p> 
7	For the <i>Button6</i> object, create a <i>direct connection</i> for displaying the just configured <i>Picture Window</i> .
8	At the <i>Button6</i> object, configure a <i>C-Action</i> that assigns a text to the button label depending whether the user is logged on or not. This <i>C-Action</i> is also triggered by the <i>@CurrentUser</i> tag.

C-Action at Button6

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
if (strcmp(GetTagChar("@CurrentUser"), ""))
return "Logoff";
else return "Logon";
}
```

- If the *@CurrentUser* tag contains a name, i.e. the comparison of the two texts results in *TRUE*, the text *Logoff* will be returned, otherwise the text *Logon* will be returned.


Note for the General Application

The following adaptations must be made before the general application:

- The names of the user groups and users, the logons and the passwords must be adapted.

3.5 Picture Zoom

An orange rectangular button with the word "Zoom" written in black text in the center.


The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_04.pdl* picture.

3.5.1 Changing the Picture Geometry between two Sizes (example 01)

Task Definition

A Picture Window is to be displayed and hidden again via two mouse-operated Buttons. When opened, the picture is to be shown small. Via another Button, the picture size is to be adjusted.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons* that will display and hide the picture in the *Smart Object* → *Picture Window* when pressed with the . Two additional *Windows Objects* → *Buttons* enlarge and reduce the picture.

Implementation in the WinCC Project

Step	Procedure: Changing the Picture Geometry between two Sizes
1	Configure a picture that is to be displayed and hidden. In the sample, the <i>pictu_3_chapter_00</i> picture (start picture of the picture project <i>Project_CreatePicture</i>) is used.
2	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> ; in the sample, this is the <i>Picture Window1</i> . Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 172 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 140. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> and the <i>Property</i> → <i>Miscellaneous</i> → <i>Adapt Picture</i> to <i>Yes</i> . In this way, the picture, which has a geometry of 859*698, is adapted to the size of the picture window. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_3_chapter_00</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> .
3	In the same picture, configure two additional <i>Windows Objects</i> → <i>Buttons</i> . In this sample, these are the <i>Button1</i> and <i>Button2</i> objects. For <i>Button1</i> , create a <i>direct connection at Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → 1 with the <i>target Object in Picture</i> → <i>Picture Window1</i> → <i>Display</i> . Apply the settings by clicking on the OK button.
4	Configure two additional <i>Windows Objects</i> → <i>Buttons</i> . In the sample, these are the <i>Button3</i> and <i>Button4</i> objects. For <i>Button3</i> , create a <i>C-Action at Events</i> → <i>Mouse</i> → <i>Press Left</i> that enlarges the <i>Picture Window</i> , hides <i>Button3</i> and displays <i>Button4</i> . For <i>Button4</i> , likewise create a <i>C-Action at Events</i> → <i>Mouse</i> → <i>Press left</i> that reduces the <i>Picture Window</i> , hides <i>Button4</i> and the displays <i>Button3</i> . Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> at both <i>Buttons</i> to <i>No</i> .

Step	Procedure: Changing the Picture Geometry between two Sizes
5	<p>For <i>Button1</i>, create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>. Connect the <i>source Constant</i> → <i>1</i> with the <i>target Object</i> in <i>Picture</i> → <i>Button3</i> → <i>Display</i>.</p> <p>Apply the settings by clicking on the <i>OK</i> button. For <i>Button2</i>, configure a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that hides <i>Button3</i> and <i>Button4</i>, reduces the size of the <i>Picture Window1</i> picture window and then hides the <i>Picture Window</i>.</p>
6	Position <i>Button3</i> and <i>Button4</i> on top of each other.

C-Action at Button3

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetHeight(lpszPictureName, "Picture Window1", 420);
SetWidth(lpszPictureName, "Picture Window1", 516);
SetVisible(lpszPictureName, "Button3", 0);
SetVisible(lpszPictureName, "Button4", 1);
}
```

- Change the height and width of the *Picture Window Picture Window1* via the *internal functions SetHeight* and *SetWidth*.
- Hide the enlargement *Button (Button3)*.
- Display the reduction *Button (Button4)*.

C-Action at Button4

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetHeight(lpszPictureName, "Picture Window1", 140);
SetWidth(lpszPictureName, "Picture Window1", 172);
SetVisible(lpszPictureName, "Button3", 1);
SetVisible(lpszPictureName, "Button4", 0);
}
```

- Change the height and width of the *Picture Window Picture Window1* via the *internal functions SetHeight* and *SetWidth*.
- Display the enlargement *Button (Button3)*.
- Hide the reduction *Button (Button4)*.

C-Action at Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetVisible(lpszPictureName, "Button3", 0);
SetVisible(lpszPictureName, "Button4", 0);
SetHeight(lpszPictureName, "Picture Window1", 140);
SetWidth(lpszPictureName, "Picture Window1", 172);
SetVisible(lpszPictureName, "Picture Window1", 0);
}
```

- Hide the enlargement button (*Button3*) and the reduction button (*Button4*).
- Change the height and width of the *Picture Window Picture Window1* via the *internal functions SetHeight* and *SetWidth*.
- Hide the *Picture Window Picture Window1*.

Note for the General Application

The following adaptations must be made before the general application:


- For the *direct connections* at the *Button1* object, the object names must be adapted.
- For the *C-Actions* at the *Button2, Button3* and *Button4* objects, the object names and the picture dimensions to be set must be adapted.

3.5.2 Changing the Picture Geometry Continuously (example 02)

Task Definition




A Picture Window is to be displayed and hidden via two mouse-operated Buttons. In addition, the size of the picture is to be made continuously adjustable via a Slider Object.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons* to display and hide the picture in the *Smart Object* → *Picture Window* when pressed with the  and a *Windows Object* → *Slider Object* to change the picture size.

Implementation in the WinCC Project

Step	Procedure: Changing the Picture Geometry Continuously
1	Configure a picture that is to be displayed and hidden. In the sample, the <i>pictu_5_window_10.pdl</i> picture is used, whose width to height ration is 2 to 1.
2	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> ; in the sample, this is the <i>Picture Window2</i> . Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 160 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 80 (width : height ratio is also 2 : 1). To have the window displayed with a border during runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> and the <i>Properties</i> → <i>Miscellaneous</i> → <i>Adapt Picture</i> to <i>Yes</i> . In this way, the picture is adapted to the size of the <i>Picture Window</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_10.pdl</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> .
3	In the same picture, configure two additional → <i>Buttons</i> . In the sample, these are the <i>Button5</i> and <i>Button6</i> objects. For <i>Button5</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → 1 with the <i>target Object in Picture</i> → <i>Picture Window2</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
4	In the same manner, create a <i>direct connection</i> for <i>Button6</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . As <i>Constant</i> , specify the value 0.
5	In Tag Management, create a tag of the <i>Unsigned 16-Bit Value</i> type. In this sample, the <i>U16i_pictu_zoom_00tag</i> is used.
6	Configure a <i>Windows Object</i> → <i>Slider Object</i> . In this sample, this is the <i>Slider Object1</i> . Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Maximum Value</i> to 300. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> to 80. At <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> , create a <i>direct connection</i> . Connect the <i>source Property</i> → <i>this object</i> → <i>Process Driver Connection</i> with the <i>target Variable</i> → <i>U16i_pictu_zoom_00</i> . Apply the settings by clicking on the <i>OK</i> button.

Step	Procedure: Changing the Picture Geometry Continuously
7	<p>For the <i>Picture Window2</i> object, create a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Geometry</i> → <i>Window Height</i>. Use the <i>Button</i>  to select the <i>tag</i> → <i>U16i_pictu_zoom_00</i>. Use the <i>Button</i>  in the <i>Change Trigger</i> dialog to confirm the <i>U16i_pictu_zoom_00</i> tag as the trigger name and set the standard cycle to <i>Upon Change</i>. Confirm the settings made by clicking on the <i>OK</i> button. In the <i>Data Type</i> field, select the <i>Direct</i> radio-button and exit the <i>Dynamic Dialog</i> by clicking on the <i>Apply</i> button.</p>
8	<p>For the <i>Picture Window2</i> object, create a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Geometry</i> → <i>Window Height</i>. The settings can be made as described above, but the <i>Expression/Formula</i> field must be completed as follow:</p> <div data-bbox="483 667 1036 793" style="border: 1px solid gray; padding: 5px; margin: 10px 0;"> <p>Expression/Formula</p> <p><input type="text" value="U16i_pictu_zoom_00*2"/> </p> </div> <p>This assigns a value of twice the window width to the window height.</p>
9	<p>For the <i>pictu_3_chapter_04</i> picture object, configure a <i>C-Action</i> at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> that sets the <i>U16i_pictu_zoom_00</i> tag to 80 when the picture is opened. Without this initialization, the value of the tag would remain at 0 until the first activation of the <i>Slider Object1</i>. If the <i>Button5</i> object would then be pressed, the <i>Picture Window2</i> would be displayed with the dimension of 0x0.</p>

C-Action for Open Picture

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPr
{
//init tag
SetTagWord("U16i_pictu_zoom_00", 80);
}
```

- Set the *U16i_pictu_zoom_00* tag to 80.

Note for the General Application

The following adaptations must be made before the general application:


- For the *direct connection* at the *Slider Object1*, the tag name must be adapted.
- For the *Dynamic Dialogs* at the *Picture Window2* object, the tag names must be adapted. The multiplier must be adapted to the width : height ratio used.

3.5.3 Configuring an adjustable Picture Geometry via the Properties Dialog (example 03)

Task Definition

A *Picture Window* is to be dragged with the mouse to any size. In addition, the picture is to be moved to any position on the screen. The picture can be maximized and hidden via a *Button*.

Implementation Concept

For the implementation, we will use two *Windows Objects* → *Buttons*, that will display and hide the picture in the *Smart Object* → *Picture Window* when pressed with the . The necessary picture properties are configured in the properties dialog box.

Implementation in the WinCC Project

Step	Procedure: Configuring an adjustable Picture Geometry via the Properties Dialog
1	Configure a picture that is to be displayed and hidden. In the sample, the pictu_3_chapter_00 picture (start picture of the picture project <i>Project_CreatePicture</i>) is used.
2	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> ; in the sample, this is the <i>Picture Window3</i> . Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 147 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 140. At <i>Properties</i> → <i>Miscellaneous</i> , set the attributes <i>Sizeable</i> , <i>Moveable</i> , <i>Border</i> , <i>Title</i> , <i>Can Be Maximized</i> , <i>Adapt Picture</i> and <i>Can Be Closed</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_3_chapter_00</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> .
3	In the same picture, configure two <i>Windows Objects</i> → <i>Buttons</i> . In this sample, the <i>Button7</i> and <i>Button8</i> objects are used. For <i>Button7</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>1</i> with the <i>target Object in Picture</i> → <i>Picture Window3</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
4	In the same manner, create a <i>direct connection</i> for <i>Button8</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . However, enter the value <i>0</i> as the <i>Constant</i> .


Note for the General Application

The following adaptations must be made before the general application:

- For the *direct connections* at the *Button7* and *Button8* objects, the picture name to be displayed and the object name of the *Picture Window* must be adapted.
- The picture displayed in the *Picture Window Picture Window3* must be adapted.

3.6 Control Windows

Operator Panels


The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_05.pdl* picture.

3.6.1 Binary Switching Operation (Two-Step Control) (example 01)

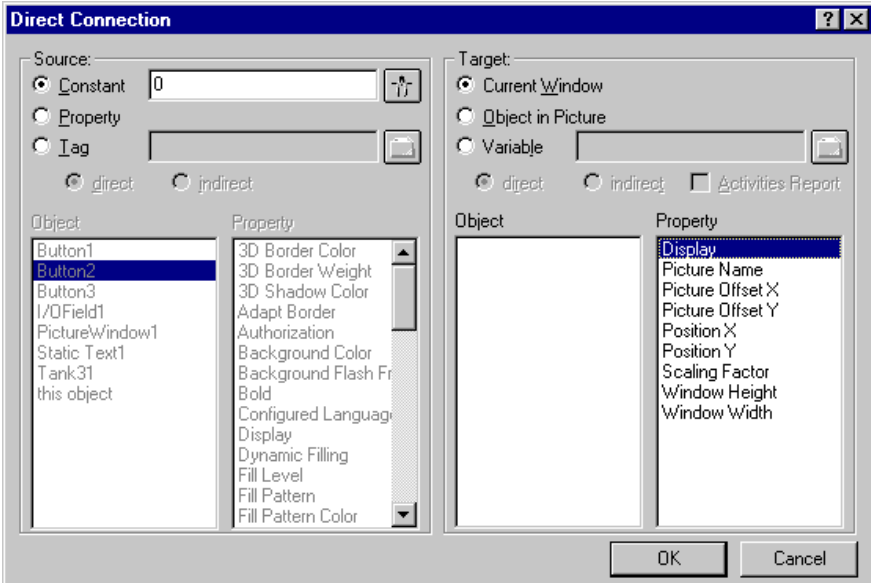
Task Definition

An operator panel is to be accessed by a mouse-operated *Button*. This operator panel is to contain a *Button* which turns a valve on and off and another *Button* to close the panel.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button* that will display the picture in a *Smart Object* → *Picture Window* when pressed with the  and two additional *Buttons* that perform the switching operation and close the panel.

Implementation in the WinCC Project

Step	Procedure: Binary Switching Operation (Two-Step Control)
1	In Tag Management, create a tag of the <i>Binary Tag</i> type. In the sample, the <i>BINi_pictu_input_00</i> tag is used. This tag contains the current status of the value.
2	<p>Configure a picture with two <i>Windows Objects</i> → <i>Buttons</i>. In the sample, the <i>pictu_5_window_11</i> picture is used containing the <i>Button1</i> and <i>Button2</i> objects. For <i>Button1</i>, create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>. Connect the <i>source Constant</i> → <i>0</i> with the target <i>Current Window</i> → <i>Display</i>. Apply the settings by clicking on the <i>OK</i> button.</p> 
3	At the second <i>Button</i> , <i>Button2</i> in our sample, configure a <i>C-Action</i> that reverses the status of the binary tag <i>BINi_pictu_input_00</i> .

Step	Procedure: Binary Switching Operation (Two-Step Control)
4	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> ; in the sample, this is the <i>Picture Window1</i> . Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 246 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 129. To display the window with a border and make it movable during runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> and the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_3_window_11.pdl</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> .
5	In the picture, configure a <i>Windows Object</i> → <i>Button</i> . In the sample, this is the <i>Button1</i> object in the <i>pictu_3_chapter_05.pdl</i> picture. For <i>Button1</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>1</i> with the target <i>Object in Picture</i> → <i>Picture Window1</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.

C-Action at Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetTagBit("BINi_pictu_input_00", (SHORT)!GetTagBit("BINi_pictu_input_00"));
}
```

- The status of the *BINi_pictu_input_00* tag is read and negated via the *internal function GetTagBit* and then reset via the *internal function SetTagBit*.

Note for the General Application

The following adaptations must be made before the general application:


- For the *direct connection* at *Button1*, the object name of the *Picture Window* to be opened must be adapted.
- For the *C-Action* at *Button2* in the operator panel, the tag name must be adapted.

3.6.2 Binary S-R Switching Operation (Two-Step Control) (example 02)

Task Definition

An operator panel is to be accessed via a mouse-operated *Button*. This operator panel is to contain a *Button* for turning a valve on and another *Button* for turning the valve off. The panel itself is to be closed by clicking on another *Button*.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button* that will display the picture in a *Smart Object* → *Picture Window* when pressed with the  and three additional *Buttons* that perform the switching operation and close the panel.

Implementation in the WinCC Project

Step	Procedure: Binary S-R Switching Operation (Two-Step Control)
1	In Tag Management, create a tag of the <i>Binary Tag</i> type. In the sample, the <i>BINi_pictu_input_01</i> tag is used. This tag contains the current status of the value.
2	Configure a picture with three <i>Windows Objects</i> → <i>Buttons</i> . In the sample, the <i>pictu_5_window_12.pdl</i> picture is used containing the <i>Button1</i> , <i>Button2</i> and <i>Button3</i> objects. For <i>Button1</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>0</i> with the target <i>Current Window</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
3	For <i>Button2</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>1</i> with the target <i>Variable</i> → <i>BINi_pictu_input_01</i> . Apply the settings by clicking on the <i>OK</i> button.
4	In the same manner, create a <i>direct connection</i> for <i>Button3</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . As <i>Constant</i> , specify the value <i>0</i> .
5	In another picture, configure a <i>Smart Object</i> → <i>Picture Window</i> ; in the sample, this is the <i>Picture Window2</i> . Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 246 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 129. To display the window with a border and make it movable during runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> and the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_3_window_12.pdl</i> picture.
6	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In the sample, this is the <i>Button2</i> object in the <i>pictu_3_chapter_05.pdl</i> picture. For <i>Button2</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>1</i> with the target <i>Object in Picture</i> → <i>Picture Window2</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.

Note for the General Application

The following adaptations must be made before the general application:


- For the *direct connection* at *Button2*, the object name of the *Picture Window* to be opened must be adapted.
- For the *direct connections* at *Button1* and *Button2* in the operator panel, the tag names must be adapted.

3.6.3 Binary Switching Operation with Acknowledgment (example 03)

Task Definition

An operator panel is to be accessed via a mouse-operated *Button*. This operator panel is to contain a *Button* for turning a valve on or off. The actual switching operation is only to take effect after pressing a separate *OK* button, which also closes the operator panel.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button* that will display the picture in a *Smart Object* → *Picture Window* when pressed with the  and two additional *Buttons* that perform the switching operation and close the panel.

Implementation in the WinCC Project

Step	Procedure: Binary Switching Operation with Acknowledgment
1	In Tag Management, create two tags of the <i>Binary Tag</i> type. In the sample, the <i>BINi_pictu_input_02</i> and <i>BINi_pictu_input_03</i> tags are used. <i>BINi_pictu_input_02</i> contains the current status of the valve, <i>BINi_pictu_input_03</i> serves as a buffer for the switching operation before its acknowledgment.
2	Configure a picture with two <i>Windows Objects</i> → <i>Buttons</i> . In the sample, the <i>pictu_5_window_13.pdl</i> picture is used containing the <i>Button1</i> and <i>Button2</i> objects. For <i>Button1</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse Action</i> . Connect the <i>source Constant</i> → <i>0</i> with the <i>target Current Window</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button. Configure another <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source BINi_pictu_input_02</i> with the <i>target BINi_pictu_input_03</i> . Apply the settings by clicking on the <i>OK</i> button.
3	For the second button, <i>Button2</i> in this sample, configure a <i>C-Action</i> that reverses the status of the binary tag <i>BINi_pictu_input_02</i> .
4	In another picture, <i>pictu_3_chapter_05.pdl</i> in this sample, configure a <i>Smart Object</i> → <i>Picture Window</i> . In the sample, this is the <i>Picture Window3</i> object. Set the <i>Property</i> → <i>Geometry</i> → <i>Width</i> to 246 and the <i>Property</i> → <i>Geometry</i> → <i>Height</i> to 129. To display the window with a border and make it movable during runtime, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Border</i> to <i>Yes</i> and the <i>Property</i> → <i>Miscellaneous</i> → <i>Moveable</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_3_window_13.pdl</i> picture.
5	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In the sample, this is the <i>Button3</i> object in the <i>pictu_3_chapter_05.pdl</i> picture. For <i>Button3</i> , create a <i>Direct Connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the <i>source Constant</i> → <i>1</i> with the <i>target Object in Picture</i> → <i>Picture Window3</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.

C-Action at Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetTagBit("BINi_pictu_input_02", (SHORT)!GetTagBit("BINi_pictu_input_02"));
}
```


- The status of the *BINi_pictu_input_02* tag is read and negated via the *internal function GetTagBit* and then reset via the *internal function SetTagBit*.

Note for the General Application

The following adaptations must be made before the general application:

- For the *direct connection* at *Button3*, the object name of the *Picture Window* to be opened must be adapted.
- For the *direct connections* at *Button1* in the operator panel, the tag names must be adapted.
- For the *C-Action* at *Button2* in the operator panel, the tag name must be adapted.


3.6.4 Automatic Input Check (example 04)

The samples pertaining to this topic are accessed in the Project_CreatePicture project by selecting the Button displayed above using the . The sample is configured in the pictu_3_chapter_05a.pdl picture.

Task Definition

An operator panel is to be accessed via a mouse-operated *Button*. This operator panel is to be used to fill a container with an amount of liquid, which is also to be entered in this panel. The value entered is to be checked automatically to determine whether it exceeds the maximum fill level of the container or not.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, that will display the picture in the *Smart Object* → *Picture Window* when pressed with the . In addition, we will use three *Windows Objects* → *Buttons* to turn the valve on and off and to close the operator panel. A *Smart Object* → *I/O Field* will be used to enter the fill level.

Implementation in the WinCC Project

Step	Procedure: Automatic Input Check
1	In Tag Management, create a tag of the <i>Binary Tag</i> type that contains the current status of the valve. In the sample, the <i>BINi_pictu_input_06</i> tag is used.
2	Create two tags of the <i>Unsigned 16-Bit Value</i> type. In this sample, these are the <i>U16i_pictu_input_04</i> and <i>U16i_pictu_input_05</i> tags. The first of these two tags contains the set-point value of the container fill level, the second the actual value.
3	Configure a picture with three <i>Windows Objects</i> → <i>Buttons</i> and a <i>Smart Object</i> → <i>I/O Field</i> . In the sample, the <i>Button1</i> , <i>Button2</i> and <i>Button3</i> as well as the <i>I/O Field1</i> objects are used. As the picture, the <i>pictu_5_window_14.pdl</i> is used.
4	In the <i>I/O Field1</i> object's <i>configuration dialog</i> , configure a <i>tag connection</i> to the <i>U16i_pictu_input_04</i> tag and trigger it upon change.
5	We assume the container has a maximum fill level of 40 liters. The <i>I/O Field</i> therefore only accepts inputs between 0 and 40. For this, set the <i>Property</i> → <i>Limits</i> → <i>Low Limit Value</i> 0 and the <i>Property</i> → <i>Limits</i> → <i>High Limit Value</i> 40.
6	For <i>Button1</i> , configure a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that hides the picture.
7	For <i>Button2</i> , configure a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that assigns the value 1 to the <i>BINi_pictu_input_06</i> tag. For <i>Button3</i> , configure a <i>direct connection</i> that assigns the value 0 to the tag.
8	In a second picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, the <i>Picture Window1</i> object is used. Adjust the dimension of the <i>Picture Window</i> to match the size of the picture just created. If the <i>Picture Window</i> is to be displayed with a border, the <i>Height</i> and <i>Width</i> of the <i>Picture Window</i> must be 10 pixels greater than those of the picture. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , select the <i>pictu_5_window_14.pdl</i> picture.

Step	Procedure: Automatic Input Check
9	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In the sample, this is the <i>Button1</i> object. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create a direct connection. Connect the source <i>Constant</i> → <i>1</i> with the target <i>Object in Picture</i> → <i>Picture Window1</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
10	To display the fill level, the <i>Tank2</i> object from the library has been used. To simulate the filling process, a <i>C-Action</i> has been created at <i>Properties</i> → <i>Geometry</i> → <i>Width</i> . At <i>Properties</i> → <i>Tag Assignment</i> → <i>Fill Level</i> , a tag connection to the <i>U16i_pictu_input_05</i> tag is configured.
11	For the second form of displaying the fill level, a <i>Smart Object</i> → <i>I/O Field</i> has been used - in the sample, this is the <i>I/O Field1</i> .

C-Action for Simulating the Filling Process

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
  BOOL state;
  SHORT level1, level2;

  //get valve state
  state=GetTagBit("BINi_pictu_input_06");

  if (state==TRUE) {
    level1=GetTagWord("U16i_pictu_input_04");
    level2=GetTagWord("U16i_pictu_input_05");
    level2++;
    if (level2>=level1) {
      SetTagBit("BINi_pictu_input_06", FALSE);
    }//if
    if (level2<=level1) {
      SetTagWord("U16i_pictu_input_05", level2);
    }//if
  }//if
  return(80);
}
```


- Reading the valve status.
- When the valve is opened, the actual and set-point values of the fill level are read. Increment the actual value. When the actual value has reached the set-point value, close the valve. Set the tag which contains the actual value.
- The return value is the width of the object.

Note for the General Application

The following adaptations must be made before the general application:

- In the *pictu_5_window_14.pdl* picture, the tag names and the limits of the *I/O Field* must be adapted to meet your requirements.


3.6.5 Enhanced Automatic Input Check (example 05)

The samples pertaining to this topic are accessed in the Project_CreatePicture project by selecting the Button displayed above using the . The sample is configured in the pictu_3_chapter_05a.pdl picture.

Task Definition

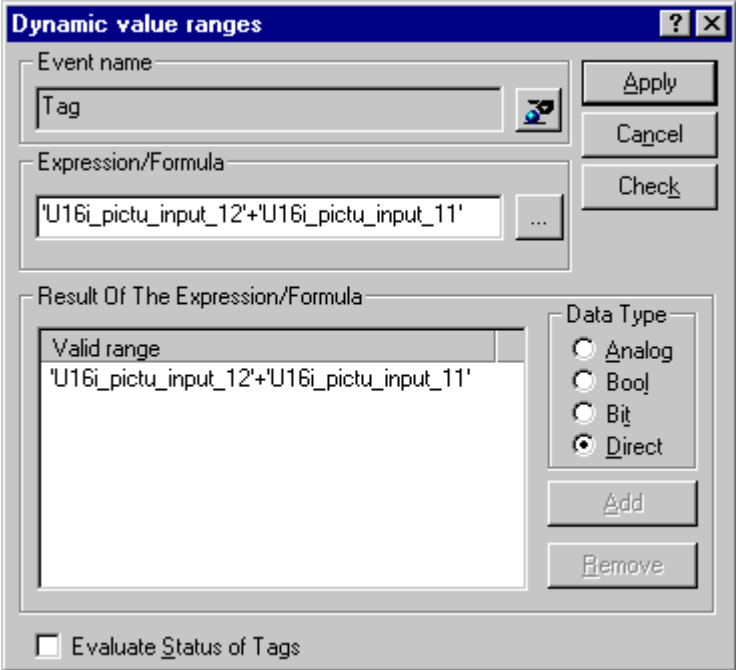
An operator panel is to be accessed via a mouse-operated *Button*. This operator panel is to be used to fill a container with two liquids in a specific ratio. The sum of the two values entered is to be checked automatically to determine whether it exceeds the maximum fill level of the container.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, that will display the picture in the *Smart Object* → *Picture Window* when pressed with the . *Three Smart Objects* → *I/O Fields* are used to enter the fill amounts. In addition, we will use two *Windows Objects* → *Buttons* to either apply the settings made in the *I/O fields* or to cancel them.

Implementation in the WinCC Project

Step	Procedure: Enhanced Automatic Input Check
1	In Tag Management, create two tags of the <i>Binary Tag</i> type that contain the current status of the valves used to fill the container. In the sample, the <i>BINi_pictu_input_09</i> and <i>BINi_pictu_input_10</i> tags are used.
2	Create four tags of the <i>Unsigned 16-Bit Value</i> type. In this sample, these are the <i>U16i_pictu_input_07</i> , <i>U16i_pictu_input_08</i> , <i>U16i_pictu_input_13</i> and <i>U16i_pictu_input_14</i> tags. The first two contain the set-point values for the container fill levels, the last two the actual values.
3	Create two tags of the <i>Unsigned 16-Bit Value</i> type. In the sample, these are the <i>U16i_pictu_input_11</i> and <i>U16i_pictu_input_12</i> tags. These contain the values entered in the <i>I/O Fields</i> .
4	Configure a picture with two <i>Windows Objects</i> → <i>Buttons</i> and three <i>Smart Objects</i> → <i>I/O Fields</i> . In the sample, the <i>Button1</i> and <i>Button2</i> as well as the <i>I/O Field1</i> , <i>I/O Field2</i> and <i>I/O Field3</i> objects are used. As the picture, <i>pictu_5_window_15.pdl</i> is used.
5	In the <i>I/O Field1</i> object's <i>configuration dialog</i> , configure a <i>tag connection</i> to the <i>U16i_pictu_input_11</i> tag and trigger it upon change. For <i>I/O Field2</i> , configure a <i>tag connection</i> to the <i>U16i_pictu_input_12</i> tag.

Step	Procedure: Enhanced Automatic Input Check
6	<p>For <i>I/O Field3</i>, configure a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i>. Enter the settings shown in the figure below. Set the trigger to upon change.</p> 
7	<p>For <i>Button2</i>, configure a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that hides the picture.</p>
8	<p>For <i>Button1</i>, create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that assigns the contents of the input tags <i>U16i_pictu_input_11</i> and <i>U16i_pictu_input_12</i> to the set-point value tags <i>U16i_pictu_input_07</i> and <i>U16i_pictu_input_08</i>. At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, configure a <i>direct connection</i> that closes the picture.</p>
9	<p>In the same picture, configure two <i>Standard Objects</i> → <i>Static Texts</i>. In the sample, the <i>Static Text5</i> and <i>Static Text6</i> objects are used. These are used to display whether the maximum fill level has been exceeded or not. At the <i>Static Text5</i> object, which contains the error message, set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i>.</p>
10	<p>For <i>I/O Field3</i>, create a <i>C-Action</i> at <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Output Value</i> that makes <i>Button1</i> operational only if the maximum fill level has not been exceeded and displays the error text if the maximum fill level has been exceeded.</p>

Step	Procedure: Enhanced Automatic Input Check
11	In a second picture, configure a <i>Smart Object</i> → <i>Picture Window</i> . In the sample, the <i>Picture Window2</i> is used. Adjust the dimension of the <i>Picture Window</i> to match the size of the picture just created. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , enter <i>pictu_5_window_15.pdl</i> .
12	In the same picture, configure a <i>Windows Object</i> → <i>Button</i> . In this sample, this is the <i>Button3</i> object. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create a direct connection. <i>Connect the source Constant</i> → <i>1</i> with the target <i>Object in Picture</i> → <i>Picture Window2</i> → <i>Display</i> . Apply the settings by clicking on the <i>OK</i> button.
13	To display the fill level, the <i>Tank2</i> object from the library has been used. To simulate the filling process, <i>C-Actions</i> have been created at <i>Properties</i> → <i>Geometry</i> → <i>Width</i> and at <i>Properties</i> → <i>Geometry</i> → <i>Height</i> . At <i>Properties</i> → <i>Tag Assignment</i> → <i>Fill Level</i> , a <i>Dynamic Dialog</i> has been created which returns the sum of the two actual value tags <i>U16i_pictu_input_13</i> and <i>U16i_pictu_input_14</i> .
14	For the second form of displaying the fill level, a <i>Smart Object</i> → <i>I/O Field</i> has been used - in the sample, this is the <i>I/O Field2</i> .

C-Action at Button1

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    SHORT tmp1, tmp2;

    tmp1=GetTagWord("U16i_pictu_input_11");
    tmp2=GetTagWord("U16i_pictu_input_12");

    if (tmp1>GetTagWord("U16i_pictu_input_07")){
        SetTagWord("U16i_pictu_input_07", tmp1);
        SetTagBit("BINi_pictu_input_09", TRUE);
    }//if
    if (tmp2>GetTagWord("U16i_pictu_input_08")){
        SetTagWord("U16i_pictu_input_08", tmp2);
        SetTagBit("BINi_pictu_input_10", TRUE);
    }//if
}
```

- Reading of the tag values that have been entered in the *I/O Fields*.
- If the value entered exceeds the current set-point value, it is transferred to the set-point value and the valve is turned on.

C-Action at I/O Field3

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char*
{
int a;

a=GetTagWord("U16i_pictu_input_11")+GetTagWord("U16i_pictu_input_12");
if (a<=40) {
    SetOperation(lpszPictureName, "Button1", 1);
    SetVisible(lpszPictureName, "Static Text5", 0);
    SetVisible(lpszPictureName, "Static Text6", 1);
} //if
else {
    SetOperation(lpszPictureName, "Button1", 0);
    SetVisible(lpszPictureName, "Static Text5", 1);
    SetVisible(lpszPictureName, "Static Text6", 0);
} //if
}
```

- Reading of the tag values that have been entered in the *I/O Fields*.
- If the sum of the values entered exceeds the maximum fill level of the container, *Button1* becomes inoperational and the *Static Text5* object containing the error message is displayed.


Note for the General Application

The following adaptations must be made before the general application:

- In the *pictu_5_window_15.pdl* picture, the tag names and the limits of the *I/O Field* must be adapted to meet your requirements .

3.6.6 Multiple Operation (example 06)


Multiple operation >>

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The sample is configured in the *pictu_3_chapter_05b.pdl* picture.

Task Definition

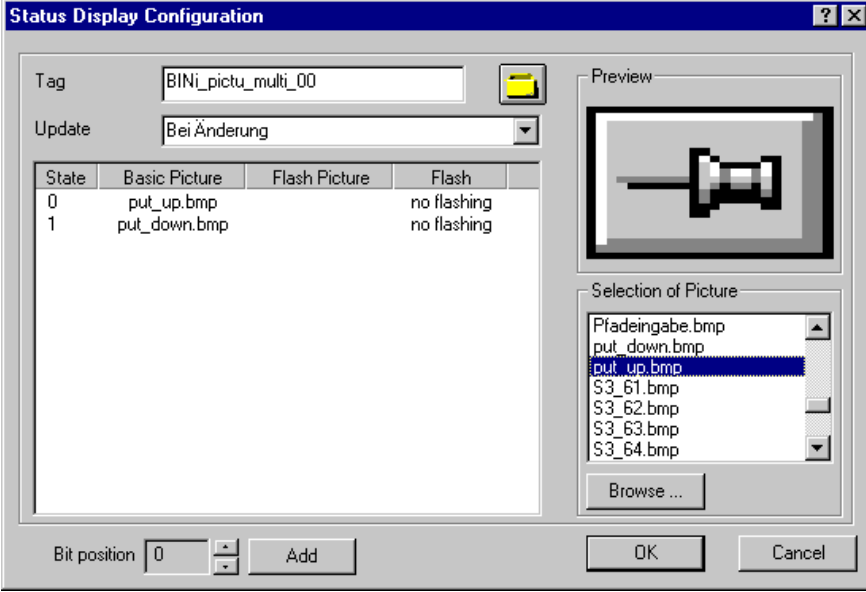
An operator panel is to be accessed via several mouse-operated *Buttons*. If the *Picture Window* is opened with a *Button*, a valve assigned to the respective *Button* can be controlled. The operator window is by default opened next to the *Button* used to call the window. It can, however, also be anchored at any other position.

Implementation Concept

For the implementation, we will use *Windows Objects* → *Buttons*, that will display the picture in the *Smart Object* → *Picture Window* when pressed with the . Two *Windows Objects* → *Buttons* are used to control the valve, and an additional button is used to close the window. The name of the valve and its status are displayed by two *Standard Objects* → *Static Texts*. The picture is anchored via a *Smart Object* → *Status Display*.

Implementation in the WinCC Project

Step	Procedure: Multiple Operation
1	In Tag Management, create tags of the <i>Binary Tag</i> type that display the current status of the valves. The number of tags required depends on the number of valves. In the sample, the <i>BINi_pictu_multi_01</i> , <i>BINi_pictu_multi_02</i> , <i>BINi_pictu_multi_03</i> and <i>BINi_pictu_multi_04</i> tags are used.
2	Create a tag of the <i>Text Tag 16-Bit Character Set</i> type. In the sample, this is the <i>T16x_pictu_input_15</i> tag. This tag will be used as an address tag.
3	Create a tag of the <i>Binary Tag</i> type. In the sample, this is the <i>BINi_pictu_multi_00</i> tag. The content of this tag contains information whether the window has been anchored.
4	Configure a picture with three <i>Windows Objects</i> → <i>Buttons</i> . In the sample, the <i>Button1</i> , <i>Button2</i> and <i>Button3</i> objects are used. As the picture, <i>pictu_5_window_16.pdl</i> is used.
5	For <i>Button1</i> , create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press left</i> that sets the position of the picture outside the visible area, closes the picture and cancels the anchoring of the picture.
6	For <i>Button2</i> , create a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> . Connect the source <i>Constant</i> → 1 with the target <i>Variable</i> → <i>T16x_pictu_input_15</i> . Select the indirect radio-button. Apply the settings by clicking on the OK button. In this way, the indirect addressing has been set. In the same manner, create a direct connection for <i>Button2</i> with the source <i>Constant</i> → 0.

Step	Procedure: Multiple Operation
7	<p>Configure a Smart Object → Status Display. In the sample, the <i>Status Display1</i> is used. In the following configuration dialog, select the <i>BINi_pictu_multi_00</i> tag and set the trigger to upon change. Use the <i>Button Add</i> to add another status. For the 0 status, select the <i>put_up.gif</i> picture and for the 1 status, the <i>put_down.gif</i> picture.</p> 
8	<p>For the <i>Status Display1</i>, create a C-Action at Events → Mouse → Press Left that negates the status of the <i>BINi_pictu_multi_00</i> tag.</p>
9	<p>For the title, configure a Standard Object → Static Text. In this sample, the <i>Static Text1</i> object is used. At Properties → Font → Text, create a C-Action that reads the current valve number from the <i>T16x_pictu_input_15</i> address tag and returns an appropriate text as the return value.</p>
10	<p>Configure another Standard Object → Static Text to display the valve status. In the sample, the <i>Static Text2</i> object is used. At Properties → Font → Text, create a C-Action that reads the status of the current valve and returns an appropriate text as the return value. At Properties → Colors → Font Color, create a C-Action that controls the font color in accordance with the status of the current valve.</p>
11	<p>In a second picture, configure a Smart Object → Picture Window. In this sample, the <i>Picture Window1</i> object is used. Adjust the dimension of the <i>Picture Window</i> to match the size of the picture just created. Set the Properties → Miscellaneous Moveable and Border to Yes. At Properties → Miscellaneous → Picture Name, set the <i>pictu_5_window_16.pdl</i> picture.</p>
12	<p>In the same picture, configure a Windows Object → Button for each valve; in the sample the <i>Button1</i>, <i>Button2</i>, <i>Button3</i> and <i>Button4</i> objects are used. Create a C-Action for each Button that reads the number of the Button and assigns the corresponding tag name to the address tag. Depending on whether the picture is anchored or not, the picture is either positioned on the right, next to the button that calls it, or not.</p>

C-Action for the Close Button (Button1)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetLeft("pictu_3_chapter_05b", "Picture Window1", -1000);
SetVisible("pictu_3_chapter_05b", "Picture Window1", 0);
SetTagBit("BINi_pictu_multi_00", FALSE);
}
```

- Set the position of the picture outside the visible area.
- Hide the picture.
- Cancel anchoring of the picture.

C-Action for Status Display1

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
SetTagBit("BINi_pictu_multi_00", (SHORT)!GetTagBit("BINi_pictu_multi_00"));
}
```

- Negate the status tag for the picture anchoring.

C-Action for the Valve Control Buttons

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
int x,y;
char name[20];
int number;
int ch = n;
char *pdest;
//check if object name contains character n
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
//read object number
else {
number = atoi(strrchr(lpszObjectName,n)+1);
sprintf(name, "BINi_pictu_multi_%02d", number);
//generate tag name
SetTagChar( "T16x_pictu_input_15", name);
}
SetVisible(lpszPictureName, "Picture Window1", 1);
if (GetTagBit("BINi_pictu_multi_UU")==FALSE){
//get object position
y=GetTop(lpszPictureName, lpszObjectName);
x=GetLeft(lpszPictureName, lpszObjectName);
//set position of picture window
SetLeft(lpszPictureName, "Picture Window1", -1000);
SetTop(lpszPictureName, "Picture Window1", y);
SetLeft(lpszPictureName, "Picture Window1", (x+22));
}
}
```

- Read the object number from the object name.
- Generate the name of the current status tag.
- Set the address tag to the current status tag.
- Display the *Picture Window*.
- If the *Picture Window* has not been anchored, determine the position of the *Button* and set the position of the picture to the right, next to the *Button*. The *Picture Window* is set outside of the visible area to avoid the picture window from being briefly displayed when its position is changed for the first time.


Note for the General Application

The following adaptations must be made before the general application:

- Adapt the object and tag names to suit your own requirements. Make sure you observe the name conventions. The button number must be uniquely assigned to the tag number to be switched.

3.7 Dynamization

Dynamics

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_06.pdl* picture.

3.7.1 Color Change (example 01)
















Task Definition

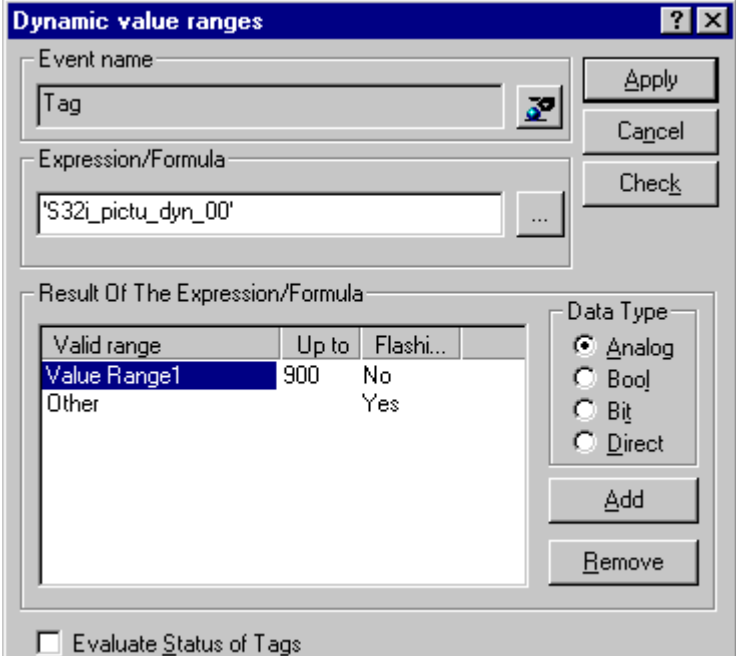
The color of a text is to change through various colors depending on the value of a tag.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Slider Object* that will change the value of a tag. The text display is implemented via a *Standard Object* → *Static Text*.

Implementation in the WinCC Project

Step	Procedure: Color Change																		
1	In Tag Management, create a tag of the <i>Signed 32-Bit Value</i> type. In the sample, this is the <i>S32i_pictu_dyn_00</i> tag.																		
2	Configure a <i>Windows Object</i> → <i>Slider Object</i> . In this sample, the <i>Slider Object1</i> is used. In the <i>configuration dialog</i> , set the <i>Maximum Value</i> to <i>1000</i> and the <i>Minimum Value</i> to <i>0</i> . At <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> , create a <i>direct connection</i> to the <i>S32i_pictu_dyn_00</i> tag.																		
3	Configure a <i>Standard Object</i> → <i>Static Text</i> . In this sample, the <i>Static Text5</i> object is used. At <i>Properties</i> → <i>Font</i> → <i>Text</i> , create a <i>C-Action</i> that outputs a text with the corresponding tag value. This C-Action is triggered upon the change of the tag.																		
4	At <i>Properties</i> → <i>Colors</i> → <i>Font Color</i> , create a <i>Dynamic Dialog</i> . In the <i>Expression/Formula</i> field, set the <i>S32i_pictu_dyn_00</i> tag and trigger it upon change. Select the <i>Data Type Analog</i> and add 4 value ranges via the <i>Add</i> button. Set the value ranges as follows: <div data-bbox="483 1283 992 1566" data-label="Table"> <table border="1"> <thead> <tr> <th>Valid range</th> <th>Up to</th> <th>Font ...</th> </tr> </thead> <tbody> <tr> <td>Value Range1</td> <td>100</td> <td></td> </tr> <tr> <td>Value Range2</td> <td>300</td> <td></td> </tr> <tr> <td>Value Range3</td> <td>600</td> <td></td> </tr> <tr> <td>Value Range4</td> <td>1000</td> <td></td> </tr> <tr> <td>Other</td> <td></td> <td></td> </tr> </tbody> </table> </div>	Valid range	Up to	Font ...	Value Range1	100		Value Range2	300		Value Range3	600		Value Range4	1000		Other		
Valid range	Up to	Font ...																	
Value Range1	100																		
Value Range2	300																		
Value Range3	600																		
Value Range4	1000																		
Other																			

Step	Procedure: Color Change
5	<p>At <i>Properties</i> → <i>Flashing</i> → <i>Flashing Background Active</i>, create a <i>Dynamic Dialog</i>. In the <i>Expression/Formula</i> field, set the <i>S32i_pictu_dyn_00</i> tag and trigger it upon change. Select the <i>Data Type Analog</i> and add a value range via the <i>Add</i> button. Set the value range as follows:</p> 

C-Action at Static Text

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    char text[100];
    DWORD temp;

    //get tag value
    temp = GetTagDWord("S32i_pictu_dyn_00");
    //generate text
    switch (GetLanguage())
    {
    case 0x407:
        sprintf(text, "Die Kesseltemperatur betragt\r\n%d Grad", temp);
        return text;
    case 0x409:
        sprintf(text, "Container Temperature is\r\n%d degree", temp);
        return text;
    case 0x40C:
        sprintf(text, "La temperature de chaudiere est\r\nde %d degre", temp);
        return text;
    default:
        sprintf(text, "Container Temperature is\r\n%d degree", temp);
        return text;
    }
}
```

- Read the tag value.
- Generate a text consisting of a text segment and a numeric value segment with the *sprintf* function. This is carried out depending on the currently set runtime language.
- The return value is the generated text.

Note for the General Application

The following adaptations must be made before the general application:

- In the *Dynamic Dialogs*, the value ranges and tag used must be adapted.

3.7.2 Text Change (example 02)

Task Definition

The texts attached to different objects are to be changed automatically depending on the status of a tag. The tool tip text is likewise to be changed.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, which is used to turn a valve on and off. A *Standard Object* → *Static Text* is used to display whether a valve is turned on or off.

Implementation in the WinCC Project

Step	Procedure: Text Change
1	In Tag Management, create a tag of the <i>Binary Tag</i> type. In this sample, the <i>BINi_pictu_dyn_01</i> tag is used.
2	Configure a <i>Windows Object</i> → <i>Button</i> . In the sample, the <i>Button1</i> object is used. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , create a <i>C-Action</i> that negates the status of the <i>BINi_pictu_dyn_01</i> tag.
3	At <i>Properties</i> → <i>Miscellaneous</i> → <i>Tooltip Text</i> , create a <i>Dynamic Dialog</i> . In the <i>Expression/Formula</i> field, set the <i>BINi_pictu_dyn_01</i> tag and trigger it upon change. Select the <i>Data Type Bool</i> and in the valid range <i>Yes/TRUE</i> , enter the text <i>close</i> and in the valid range <i>No/FALSE</i> , enter the text <i>open</i> .
4	Configure a <i>Standard Object</i> → <i>Static Text</i> . In this sample, the <i>Static Text7</i> object is used. At <i>Properties</i> → <i>Font</i> → <i>Text</i> , create a <i>Dynamic Dialog</i> . In the <i>Expression/Formula</i> field, set the <i>BINi_pictu_dyn_01</i> tag and trigger it upon change. Select the <i>Data Type Bool</i> and in the valid range <i>Yes/TRUE</i> , enter the text <i>valve opened</i> and in the valid range <i>No/FALSE</i> , enter the text <i>valve closed</i> .

Note for the General Application

The following adaptations must be made before the general application:

- In the *Dynamic Dialogs*, the texts and the tag used must be adapted to meet your own requirements.

3.7.3 Animation of Movement (example 03)

Task Definition

An object is to be moved to a specific position on the screen depending on a tag value.

Implementation Concept

For the implementation, we will use a *Smart Object* → *Picture Window*, whose position is controlled by a tag. A *Windows Object* → *Slider Object* is used to change the values of the tag.

Implementation in the WinCC Project

Step	Procedure: Animation of Movement
1	In Tag Management, create a tag of the <i>Signed 32-Bit Value</i> type. In this sample, the <i>S32i_pictu_dyn_03</i> tag is used.
2	Configure a <i>Windows Object</i> → <i>Slider Object</i> ; in this sample, the <i>Slider Object2</i> is used. In the <i>configuration dialog</i> , set the <i>Maximum Value</i> to 300 and the <i>Minimum Value</i> to 0. At <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> , create a <i>direct connection</i> to the <i>S32i_pictu_dyn_03</i> tag.
3	Configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, the <i>Picture Window1</i> object is used. Set the <i>Properties</i> → <i>Miscellaneous Border</i> and <i>Adapt Picture</i> to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , set the <i>pictu_3_chapter_00.pdl</i> picture.
4	At <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> , create a <i>Dynamic Dialog</i> . In the <i>Expression/Formula</i> field, enter the expression $((S32i_pictu_dyn_03*2)+90)$. Set the trigger to upon change of the <i>S32i_pictu_dyn_03</i> tag. Select the <i>Data Type Direct</i> .
5	At <i>Properties</i> → <i>Geometry</i> → <i>Position Y</i> , create a <i>Dynamic Dialog</i> . In the <i>Expression/Formula</i> field, enter the expression $(400-S32i_pictu_dyn_03)$. Set the trigger to upon change of the <i>S32i_pictu_dyn_03</i> tag. Select the <i>Data Type Direct</i> .

Note for the General Application

The following adaptations must be made before the general application:

- In the *Dynamic Dialogs*, the expressions for calculating the bit position must be adapted to meet your own requirements.
- The tag name must also be adapted.

3.7.4 Displaying and Hiding Objects using a Bit Evaluation (example 04)


Task Definition

Objects are to be displayed and hidden depending on a specific bit position in a tag value.

Implementation Concept

For the implementation, a *Windows Object* → *Check-Box* is used, which sets the individual Bits of a tag. A number of *Standard Objects* → *Polygons* are displayed or hidden, depending on these bits.

Implementation in the WinCC Project

Step	Procedure: Displaying and Hiding Objects using a Bit Evaluation
1	In Tag Management, create a tag of the <i>Unsigned 8-Bit Value</i> type. In this sample, the U08_pictu_dyn_02 tag is used.
2	Configure a <i>Windows Object</i> → <i>Check-Box</i> ; in this sample the <i>Check-Box1</i> object is used. At <i>Properties</i> → <i>Geometry</i> → <i>Number of Boxes</i> , set the number of objects to be switched; in the sample, this is 7. At <i>Properties</i> → <i>Font</i> → <i>Text</i> , enter the name of the object that is to be switched by the corresponding bit for each index value. At <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Selected Boxes</i> → <i>Change</i> , create a direct connection with the source <i>Property</i> → <i>Check-Box 1</i> → <i>Selected Boxes</i> and the target <i>Variable U08i_pictu_dyn_02</i> .
3	Configure several <i>Standard Objects</i> → <i>Polygon</i> . In this sample, the <i>Polygon1 to Polygon7</i> objects are used.
4	For the <i>Polygon1</i> object, create a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i> . In the <i>Expression/Formula</i> field, set the <i>U08i_pictu_dyn_02</i> tag and trigger it upon change. Select the <i>Data Type Bit</i> . Use the  button to open the bit selection dialog and select the first bit. <div data-bbox="521 1379 1115 1705" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Bit selection ✕</p> <p>Bit dialog <input type="text" value="0x1"/> <input type="button" value="OK"/></p> <p>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16</p> <p>○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○</p> <p>○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○</p> <p>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> </div>
5	Proceed in the same manner for the remaining <i>Polygon</i> objects, but change the bit number of each one.


Note for the General Application

The following adaptations must be made before the general application:

- In the *Dynamic Dialogs*, the tag names and the picture position must be adapted to meet your own requirements.

3.7.5 Animation of Movement via a C-Action (example 05)



The following samples of the Adding Dynamics chapter can be accessed in the *Project_CreatePicture* project by clicking on the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_06a.pdl* picture.


Task Definition

An object is to be moved in one direction by clicking on a *Button* and in another direction by clicking on another *Button*.

Implementation Concept

For the implementation, we will use a *Smart Object* → *Status Display* to display two pictures. Two *Windows Objects* → *Buttons* are used to move this status display in two different directions.

Implementation in the WinCC Project

Step	Procedure: Animation of Movement via a C-Action
1	In Tag Management, create three tags of the <i>Binary Tag</i> type; in this sample, the BINi_pictu_dyn_05, BINi_pictu_dyn_06 and BINi_pictu_dyn_07 tags are used.
2	Configure a <i>Smart Object</i> → <i>Status Display</i> . In this sample, the Status Display1 object is used. In the <i>configuration dialog</i> , set the BINi_pictu_dyn_05 tag and the trigger to upon change. Add another status. For the status 0, set the <i>Ferrari1.gif</i> picture and for the status 1, set the <i>Ferrari2.gif</i> picture. 
3	At <i>Properties</i> → <i>State</i> → <i>Basic Picture Transparent Color</i> , set the color <i>White</i> for both states (1 and 0) and set the <i>Picture Transparent Color On</i> to <i>Yes</i> . This means that the picture is not shown with a white background.
4	Configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button1</i> is used. At <i>Events</i> → <i>Mouse</i> → <i>Press left</i> , create a <i>direct connection</i> that sets the BINi_pictu_dyn_07 tag to 1 and at <i>Events</i> → <i>Mouse</i> → <i>Press right</i> , create a <i>direct connection</i> that resets the same tag to 0.
5	At a second <i>Windows Object</i> → <i>Button</i> , create two <i>direct connections</i> to the Variable BINi_pictu_dyn_06 tag in the same manner as outlined above. In this sample, the <i>Button2</i> object is used.
6	For the <i>Status Display1</i> object, create a <i>C-Action</i> at <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> that executes the animation of movement depending on which <i>Button</i> is pressed. Set the trigger of this action to 250 ms.

C-Action for Animation of Movement

```

#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN
{
static int a = 90;

//forward
if (GetTagBit("BINi_pictu_dyn_07")&&(a<652)) {
    a+=20;
    SetTagBit("BINi_pictu_dyn_05", (SHORT)!GetTagBit("BINi_pictu_dyn_05"));
}
//rewind
if (GetTagBit("BINi_pictu_dyn_06")&&(a>-0)) {
    a-=10;
    SetTagBit("BINi_pictu_dyn_05", (SHORT)!GetTagBit("BINi_pictu_dyn_05"));
}
//return x-position
return a;
}

```

- Define a tag of the *static int* type and initialize it with the current X position of the object.
- Check whether *Button1* is pressed and whether the X position is smaller than 652. If yes, increase the value that contains the X position by 20. Then change the picture displayed in the *Status Display1*.
- Check whether *Button2* is pressed and whether the X position is greater than -200. If yes, decrease the value that contains the X position by 10. Then change the picture displayed in the *Status Display1*.
- The return value is the new X position.

Note for the General Application

The following adaptations must be made before the general application:

- The principle of the animation can be transferred.

3.7.6 Creating Animation of Movement with a Wizard (example 06)


Task Definition

An object is to change its position on the screen when changes are made to a tag. Separate tags are to be used for the X and Y positions. Configuration is carried out via the *Dynamic Wizard*.

Implementation Concept

For the implementation, we will use a *Standard Object* → *Circle*, which is to be moved on screen. For the tag input, two *Windows Objects* → *Slider Objects* are used.

Implementation in the WinCC Project

Step	Procedure: Creating Animation of Movement with a Wizard
1	In Tag Management, create two tags of the <i>Unsigned 32-Bit Value</i> type. In this sample, the S32i_pictu_dyn_10 and S32i_pictu_dyn_11 tags are used.
2	Configure two <i>Windows Objects</i> → <i>Slider Objects</i> ; in this sample, <i>Slider Object1</i> and <i>Slider Object2</i> are used. For <i>Slider Object1</i> , create a <i>direct connection</i> . Connect the <i>source Properties</i> → <i>Slider Object1</i> → <i>Process Driver Connection</i> with the Variable S32i_pictu_dyn_10. In the same manner, create a <i>direct connection</i> for <i>Slider Object2</i> to the Variable S32i_pictu_dyn_11.
3	In the <i>configuration dialogs</i> of the <i>Slider Objects</i> , set the <i>Maximum Value</i> to 255.
4	Configure a <i>Standard Object</i> → <i>Circle</i> . In this sample, the <i>Circle1</i> object is used. While the object is highlighted, select the <i>Standard Dynamics</i> tab and then the <i>Move Object</i> entry from the <i>Dynamic Wizard</i> via a  . As the trigger, select <i>Tag</i> . On the <i>Set Options</i> page, select the S32i_pictu_dyn_10 tag for the X direction and the S32i_pictu_dyn_11 tag for the Y direction. Enter 0 and 255 respectively as the low and high limit for formatting. On the next page, specify the picture area within which the object is to be moved. Click on <i>Finish</i> to complete the Wizard.
5	In the <i>C-Actions</i> generated by the <i>Dynamic Wizard</i> , set the trigger to upon change for the corresponding tag used at <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> and at <i>Properties</i> → <i>Geometry</i> → <i>Position Y</i> .

C-Action generated by the Wizard at Position X

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN
{
long i,j,k;
i=GetTagWord("S32i_pictu_dyn_10");
j=((i-0)*100/(255-0));
k=min(((j*(690-490))/100)+490,690);
return max(490,k);
}
```

Note for the General Application

The following adaptations must be made before the general application:

- The settings made in the *Dynamic Wizard* for the animation of movement must be adapted to meet your own requirements.

3.7.7 Color Change via a C-Action (example 06)

Task Definition

The color of an object is to change smoothly from a dark to a light shade as a tag value changes.

Implementation Concept

For the implementation, we will use a Standard Object → Circle, whose color changes as a tag value changes. To input a tag value, a Windows Object → Slider Object is used.

Implementation in the WinCC Project

Step	Procedure: Color Change via a C-Action
1	In Tag Management, create a tag of the <i>Unsigned 32-Bit Value</i> type; in this sample, the <i>S32i_pictu_dyn_10</i> tag is used.
2	Configure a <i>Windows Object</i> → <i>Slider Object</i> . In this sample, the <i>Slider Object1</i> is used. At <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> of the <i>Slider Object1</i> , create a <i>direct connection</i> . Connect the <i>source Properties</i> → <i>Slider Object1</i> → <i>Process Driver Connection</i> with the Variable <i>S32i_pictu_dyn_10</i> .
3	At the <i>Slider Object1</i> , set the <i>Property</i> → <i>Miscellaneous</i> → <i>Maximum Value</i> to 255.
4	Configure a <i>Standard Object</i> → <i>Circle</i> ; in this sample, the <i>Circle1</i> object is used. At <i>Properties</i> → <i>Colors</i> → <i>Background Color</i> , create a <i>C-Action</i> that supplies a color value depending on the <i>S32i_pictu_dyn_10</i> tag. This action is triggered upon change of this tag.

C-Action for the Color Change

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN
{
return (GetTagDWord("S32i_pictu_dyn_10")<<8);
}
```

- The action returns the as the return value the *S32i_pictu_dyn_10* tag read shifted to the left by 8 bit positions.

Note for the General Application

The following adaptations must be made before the general application:

- The color values are coded by specifying values for red, green and blue. 8 bits are reserved for each of these values in the 24-Bit color value. In this sample, the tag value has been shifted 8 bits to the left and therefore represents the green value. If this is not done, the color will change from black to red; if the tag is shifted 16 bits, from black to blue.

3.7.8 Animation of Movement via a Status Display (example 07)

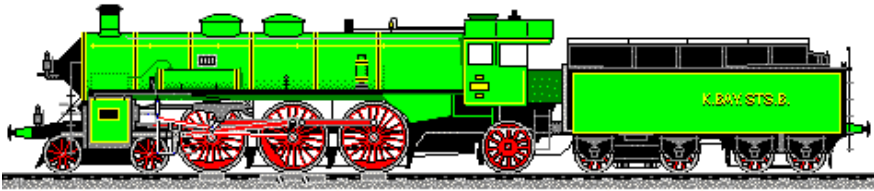
Task Definition

Movement is to be simulated by turning on different pictures in a *Smart Object* → *Status Display*.

Implementation Concept

For the implementation, we will use a *Smart Object* → *Status Display*, in which we display different pictures one after the other after turning on the display by means of another *Smart Object* → *Status Display*.

Implementation in the WinCC Project

Step	Procedure: Animation of Movement via a Status Display
1	In Tag Management, create a tag of the <i>Binary Tag</i> type. In this sample, the BINi_pictu_dyn_09 tag is used.
2	Configure a <i>Smart Object</i> → <i>Status Display</i> . In this sample, the <i>Status Display3</i> object is used. In the <i>configuration dialog</i> , set the <i>BINi_pictu_dyn_09 tag</i> and the trigger to <i>upon change</i> . Add another <i>status</i> . For the <i>status 0</i> , set the <i>Smili.gif</i> picture and for the <i>status 1</i> the <i>Ohh.gif</i> picture.
3	For the <i>Status Display3</i> object, create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that negates the status of the <i>BINi_pictu_dyn_09 tag</i> .
4	Configure another <i>Smart Object</i> → <i>Status Display</i> , in this sample, the <i>Status Display4</i> object is used. Via <i>Properties</i> → <i>State</i> → <i>Current Status</i> , add seven additional stati with the corresponding pictures. For each status, set the <i>Property Basic Picture Transparent Color</i> to <i>White</i> and the <i>Property Basic Picture Transparent Color On</i> to <i>Yes</i> . The stati from 0 to 7 are each assigned one of the pictures from <i>S3_61.gif</i> to <i>S3_68.gif</i> .
	
5	For the <i>Status Display4</i> object, create a <i>C-Action</i> at <i>Properties</i> → <i>State</i> → <i>Current Status</i> that initiates the run of the current stati 0 to 7. Set the trigger for this action to <i>250 ms</i> .

C-Action for Status Display4

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN
{
static int a = 0, b = 0;
if (GetTagBit("BINi_pictu_dyn_09")) {
    if (b==0) a++;
    else a--;
    if (a==7) b=1;
    if (a==0) b=0;
}
return a;
}
```

- Declare two tags of the *static int* type and initialize them with zero.
- If the animation is activated, run through tags 0 through 7 and then begin at 0 again.
- Return this tag as the return value.

Note for the General Application


The following adaptations must be made before the general application:

- The principle of the animation can be transferred.
- The *Status Display3* object can be integrated into other projects in the form of a switch object if the status pictures and the tag name are adapted.

3.8 Language Switch



Language

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_07.pdl* picture.

3.8.1 Runtime Language Switch (example 01)



Task Definition

The runtime language is to be changed via one button assigned to each language set.

Implementation Concept

For the implementation, we will use three *Windows Objects* → *Buttons* that we can take from the Library completely configured.

Implementation in the WinCC Project

Step	Procedure: Runtime Language Switch
1	In the <i>Graphics Designer</i> , configure any picture in a certain language. Via the <i>View</i> → <i>Language</i> menus, the next language to be configured is selected and all the texts are translated into this language. Using the <i>language.exe</i> program located on the WinCC CD-ROM, all texts used in a project can be exported as a <i>csv file</i> . They can then be translated and imported back into the project.
2	Open the library via the <i>View</i> → <i>Library</i> menus. From the folder <i>Global Library</i> → <i>Buttons Language</i> , select the corresponding <i>Buttons</i> . This is most easily accomplished by clicking on the desired object and dragging it to the work field while keeping the  pressed.
3	If a language not part of the library is required, a <i>C-Action</i> for a <i>Windows Object</i> → <i>Button</i> must be created at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that performs the language switch to the corresponding language. A <i>Dynamic Wizard</i> is also available to generate the appropriate <i>C-Action</i> . To apply the Wizard, highlight the <i>Button</i> , select the <i>System Functions</i> tab and then the <i>Language Switch</i> entry via a  from the <i>Dynamic Wizard</i> . In the <i>Dynamic Wizard</i> , the desired language can be selected.

C-Action for the German Button

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
SetLanguage(0x407); //Rückgabe-Typ :BOOL
}
```

- Use the *SetLanguage* function to change the language setting by entering the corresponding language code.

Note for the General Application

The following adaptations must be made before the general application:

- In the *Dynamic Wizard*, make the desired language settings.

3.8.2 Dialog Box for the Runtime and Control Center Language Switch (example 02)

Task Definition

Via a *Button*, a dialog box is to be called in which one of the set languages can be selected.

Implementation Concept

For the implementation, we will use a *Windows Object* → *Button*, which displays or hides a *Smart Object* → *Picture Window*. The dialog box can be taken directly from the *Project_CreatePicture* project.

Implementation in the WinCC Project

Step	Procedure: Dialog Box for the Runtime and Control Center Language Switch
1	In the <i>Graphics Designer</i> , configure any picture in a certain language. Via the <i>View</i> → <i>Language</i> menus, the next language to be configured is selected and all the texts are translated into this language.
2	Configure a <i>Smart Object</i> → <i>Picture Window</i> . In this sample, the <i>Picture Window1</i> object is used. At <i>Properties</i> → <i>Geometry</i> , set the <i>Window Height</i> to 230 and the <i>Window Width</i> to 214. At <i>Properties</i> → <i>Miscellaneous</i> , set the <i>Moveable</i> , <i>Border</i> , <i>Title</i> and <i>Can Be Closed</i> entries to <i>Yes</i> . At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , set the <i>pictu_5_window_19.pdl</i> picture. This picture is located in the <i>Project_CreatePicture</i> project and can be used without making changes. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i> .
3	Configure a <i>Windows Object</i> → <i>Button</i> . In this sample, the <i>Button4</i> object is used. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> , configure a <i>direct connection</i> that makes the <i>Picture Window1</i> object visible.


Note for the General Application

The following adaptations must be made before the general application:

- The *pictu_5_window_19.pdl* picture can be reused in another project without having to make changes.

3.9 Operation without a Mouse

**Cursorcontrol**

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_08.pdl*, *pictu_3_chapter_08a.pdl* and *pictu_3_chapter_08b.pdl* pictures.

3.9.1 Operation via TAB Key or Hotkey (example 01)

Task Definition

A text is to be formatted using various dialogs. Its font color and various font properties, e.g. font size, are to be set. In addition, the settings made are to be resettable to the default settings.



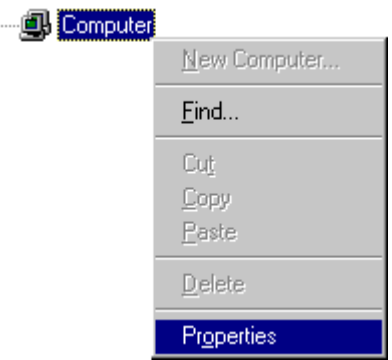

The operation of all elements in the picture is to be carried out exclusively via the keyboard.

Implementation Concept

For the implementation, we will use four *Windows Objects* → *Buttons*. They make the dialogs visible. The dialogs can be operated with the keyboard, if the runtime cursor has been turned on. The selection of the button to be operated is carried out via the TAB key. In addition, a hotkey is assigned to each button.

To display the dialogs, three *Smart Objects* → *Picture Windows* are used.

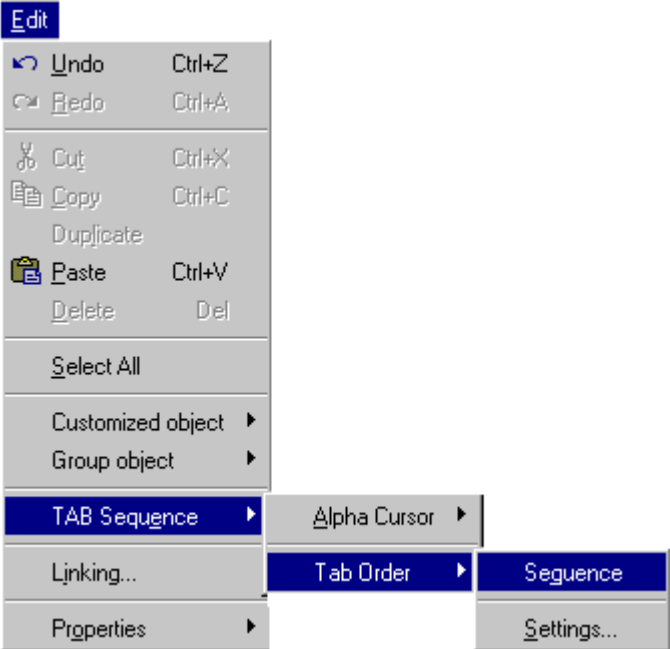
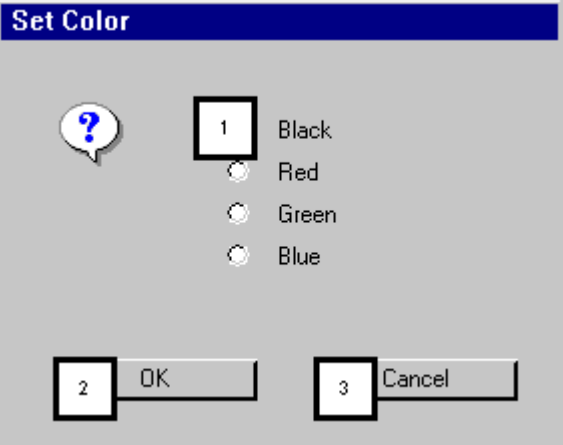
Configuring the Cursor Control

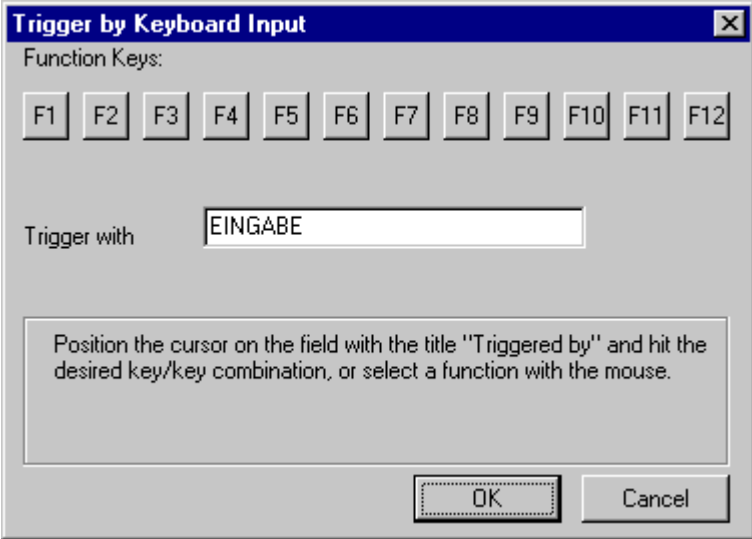
Step	Procedure: Configuring the Cursor Control
1	<p>In the <i>Control Center</i>, make the settings for the Cursor Control.  R on the <i>Computer</i> entry and then select <i>Properties</i> from the pop-up menu. In the following <i>Computer List Properties</i> dialog, click on the  <i>Button Properties</i>.</p>  <p>Select the <i>Graphics Runtime</i> tab.</p>
2	<p>Make the hotkey settings as follows. For the <i>Window on Top</i> command, no hotkey is configured, since in the samples the operating focus is set via <i>C-Actions</i>.</p> <p>To switch between the <i>Tab Order/Alpha Cursor</i>, set <i>SHIFT+A</i> and to turn the Runtime Cursors On/Off, set <i>SHIFT+R</i>.</p> 

Step	Procedure: Configuring the Cursor Control
3	In the <i>Cursor Control: Keys</i> field, no keys need to be set. If they are required in the samples, they will be set using an API function. This is done to demonstrate various operating concepts in the samples. In normal cases, a certain operating concept is selected for a project and set in here.

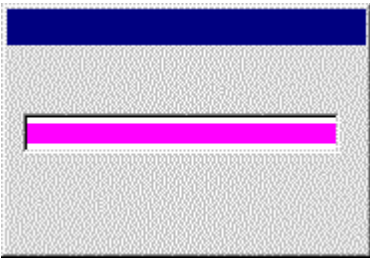


Implementation in the WinCC Project

Step	Procedure: Configuring the Hotkey Operation
1	In Tag Management, create three tags of the <i>Unsigned 16-Bit Value</i> type that will contain the font properties set. In this sample, the <i>U16i_pictu_cursor_00</i> to <i>U16i_pictu_cursor_02</i> tags are used.
2	In the <i>pictu_3_chapter_08.pdl</i> picture, configure four objects of the <i>Windows Object</i> → <i>Button</i> type. In this sample, these are the <i>Button1</i> , <i>Button2</i> , <i>Button3</i> and <i>Button4</i> objects. They used to display the dialogs and to reset the settings made. In addition, configure a <i>Standard Object</i> → <i>Static Text</i> , whose font properties are set via the dialogs. In this sample, the <i>Static Text1</i> object is used.
3	Configure another picture which serves as the dialog for setting the color. In this sample, this is the <i>pictu_5_window_23.pdl</i> picture. In this picture, configure a <i>Windows Object</i> → <i>Option Group</i> . In this sample, this is the <i>Option Group1</i> object. Set the <i>Property</i> → <i>Geometry</i> → <i>Number of Boxes</i> to 4. This enables the selection of four different colors. At <i>Properties</i> → <i>Output/Input</i> → <i>Selected Boxes</i> , create a <i>tag connection</i> to the <i>U16i_pictu_cursor_00</i> tag. At <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> , create a <i>C-Action</i> that sets the focus to object just created. This <i>C-Action</i> is triggered every 1 h. If the <i>Option Group</i> has the operating focus while the runtime cursor is active, it is displayed with a border. If rectangles of the same color as the background are placed on top of the border, the border of the object can be made invisible.
4	In the same picture, configure two <i>Windows Objects</i> → <i>Buttons</i> . In the sample, these are the <i>Button1</i> and <i>Button2</i> objects. <i>Button1</i> is used as the <i>OK Button</i> . At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> , create a <i>C-Action</i> that changes the color of the text depending on the value of the <i>U16i_pictu_cursor_00</i> tag and then hides this dialog. <i>Button2</i> is used as the <i>Cancel Button</i> . At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> , create a <i>direct connection</i> that makes the window invisible.

Step	Procedure: Configuring the Hotkey Operation
5	<p>Make the settings for the keyboard operation.</p> <p>Set the Tab order. This is done via the <i>Edit</i> → <i>TAB Sequence</i> → <i>Tab Order</i> → <i>Sequence</i> menus.</p>  <p>The screenshot shows the 'Edit' menu with the following items: Undo (Ctrl+Z), Redo (Ctrl+Y), Cut (Ctrl+X), Copy (Ctrl+C), Duplicate, Paste (Ctrl+V), Delete (Del), Select All, Customized object, Group object, TAB Sequence, Linking..., and Properties. The 'TAB Sequence' menu is open, showing 'Alpha Cursor' and 'Tab Order'. The 'Tab Order' menu is open, showing 'Sequence' and 'Settings...'.</p> <p>Each object that can be operated is now displayed with a number. The order of the numbers represents the tab order. This is the order in which the objects are accessed when the TAB key is pressed. Using the mouse, the order can be changed by clicking on the individual numbers.</p> <p>The order is set as follows:</p>  <p>The screenshot shows the 'Set Color' dialog box with a question mark icon and a list of objects: 1 (Black), 2 (Red), 3 (Green), and 4 (Blue). The 'OK' button is labeled '2' and the 'Cancel' button is labeled '3'.</p>

Step	Procedure: Configuring the Hotkey Operation
	<p>The selection in the <i>Option Group</i> is performed via the <i>cursor keys</i>. The selection of a color is performed via the <i>spacebar</i>. The <i>TAB</i> key is used to switch among the operating elements. The buttons are operated via the <i>space bar</i>.</p> <p>In addition, hotkeys are assigned to both <i>Buttons</i>. Via <i>Properties</i> → <i>Miscellaneous</i> → <i>Hot Key</i>, the dialog for the configuration of a hotkey is opened. For the <i>OK Button</i>, the <i>ENTER</i> key is set, for the <i>Cancel Button</i>, the <i>ESC</i> key is set.</p> 
6	<p>In the <i>pictu_3_chapter_08.pdl</i> picture, configure a <i>Smart Object</i> → <i>Picture Window</i>, in which the picture just configured is displayed. In this sample, this is the <i>Picture Window1</i> object. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the <i>pictu_3_window_23.pdl</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display to No</i>.</p>
7	<p>For <i>Button1</i>, create a <i>C-Action</i> that queries the currently set color of the text and writes to the <i>U16i_pictu_cursor_00</i> tag depending on the result. This is done to set the selection in the <i>Option Group</i> of the dialog to the currently set value. In addition, the <i>Picture Window1</i> object is displayed.</p> <p>For <i>Button1</i>, create a <i>C-Action</i> at <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> that sets the operating focus to this object. This <i>C-Action</i> is triggered every <i>1 h</i>, however the focus is only set at the first time.</p> <p>Set Color F9</p> <p>In addition, a hotkey is assigned to this <i>Button</i>. In the sample, this is the <i>F9</i> function key.</p>

Step	Procedure: Configuring the Hotkey Operation
8	<p>Configure a picture that will serve as the dialog for setting various font properties. In the sample, this is the <i>pictu_5_window_24.pdl</i> picture.</p> <p>In this picture, configure a <i>Windows Object</i> → <i>Check-Box</i>. In this sample, the <i>Check-Box1</i> object is used. Set the <i>Property</i> → <i>Geometry</i> → <i>Number of Boxes</i> to 4. The selection of the <i>Bold</i>, <i>Italic</i>, <i>Underline</i> and <i>Border</i> properties is to be made possible.</p> <p>At <i>Properties</i> → <i>Output/Input</i> → <i>Selected Boxes</i>, create a <i>tag connection</i> to the <i>U16i_pictu_cursor_01</i> tag.</p> <p>At <i>Properties</i> → <i>Geometry</i> → <i>Position X</i>, create a <i>C-Action</i> that sets the focus to this object. The selection frame can be hidden using the same procedure as described for the <i>Option Group</i>.</p> <p>Just like for the <i>pictu_5_window_23.pdl</i> picture, configure two <i>Windows Objects</i> → <i>Buttons</i>. If the <i>OK Button</i> is activated, the <i>U16i_pictu_cursor_01</i> tag is read and the corresponding settings are applied to the text.</p> <p>The settings pertaining to the keyboard operation are made in the same manner as for the <i>pictu_5_window_23.pdl</i> picture.</p>
9	<p>In the <i>pictu_3_chapter_08.pdl</i> picture, configure another <i>Smart Object</i> → <i>Picture Window</i>, in which the picture just configured is displayed. In this sample, this is the <i>Picture Window2</i> object. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the <i>pictu_3_window_24.pdl</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display to No</i>.</p>
10	<p>For <i>Button2</i>, create a <i>C-Action</i> that queries the currently set font properties to be changed and writes them to the <i>U16i_pictu_cursor_01</i> tag depending on the result. This is done to set the selections in the <i>Check-Box</i> of the dialog to the currently set values. In addition, the <i>Picture Window2</i> object is displayed.</p> <p>Furthermore, a hotkey is assigned to this <i>Button</i>. In the sample, this is the <i>F10</i> function key.</p> <p>Format F10</p>

Step	Procedure: Configuring the Hotkey Operation
11	<p>Configure another picture that will serve as the dialog for setting the font size. In this sample, this is the <i>pictu_5_window_25.pdl</i> picture.</p> <p>In this picture, configure a <i>Smart Object</i> → <i>I/O Field</i>. In this sample, the <i>I/O Field1</i> object is used.</p> <p>At <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i>, create a <i>tag connection</i> to the <i>U16i_pictu_cursor_02</i> tag.</p> <p>At <i>Properties</i> → <i>Geometry</i> → <i>Position X</i>, create a <i>C-Action</i> that sets the focus to this object. The selection frame is hidden by positioning a <i>Graphic Object</i> on top of the <i>I/O Field</i>. In this sample, the <i>Graphic Object1</i> is used. The bitmap displayed by the <i>Graphic Object</i> has a certain color in the area where the <i>I/O Field</i> is to be displayed. This color in the <i>Graphic Object</i> is set at <i>Properties</i> → <i>Picture</i> → <i>Picture Transparent Color</i>. Additionally, the <i>Property</i> → <i>Picture</i> → <i>Picture Transparent Color On</i> is set to <i>Yes</i>. The following displays the bitmap used.</p> 
12	<p>Just like for the <i>pictu_5_window_23.pdl</i> picture, configure two <i>Windows Objects</i> → <i>Buttons</i>. If the <i>OK Button</i> or the <i>Cancel Button</i> is pressed, the dialog is closed.</p> <p>Both <i>Buttons</i>, however, are excluded from the TAB order. This is done via the <i>Edit</i> → <i>TAB Sequence</i> → <i>Tab Order</i> → <i>Sequence</i> menus. An object can be removed from the TAB order by pressing and holding the <i>CTRL+SHIFT</i> key combination and selecting it with the . Instead of the number, a * is displayed in the white rectangle.</p>  <p>The operation of the <i>Buttons</i> is to be performed exclusively via the hotkeys <i>ENTER</i> and <i>ESC</i>. If the <i>ENTER</i> key is pressed, the value entered in the <i>I/O Field</i> is transferred to the <i>U16i_pictu_cursor_02</i> tag.</p> <p>For the <i>Static Text1</i> object in the <i>pictu_3_chapter_08.pdl</i> picture, create a <i>tag connection</i> at <i>Properties</i> → <i>Font</i> → <i>Font Size</i> to the <i>U16i_pictu_cursor_02</i> tag.</p>
13	<p>In the <i>pictu_3_chapter_08.pdl</i> picture, configure another <i>Smart Object</i> → <i>Picture Window</i>, in which the picture just configured is displayed. In this sample, this is the <i>Picture Window3</i> object. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the <i>pictu_3_window_25.pdl</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i>.</p>

Step	Procedure: Configuring the Hotkey Operation
14	<p>For <i>Button3</i>, create a <i>C-Action</i> that displays the <i>Picture Window3</i> object. In addition, a hotkey is assigned to this <i>Button</i>. In the sample, this is the <i>F11</i> function key.</p> <p>Font Size F11</p>
15	<p>For <i>Button4</i>, create a <i>C-Action</i> that resets the setable properties of the <i>Static Text1</i> object to the default values. The <i>Button</i> is assigned the hotkey <i>F12</i>.</p> <p>Reset F12</p>
16	<p>Via the <i>Edit</i> → <i>TAB Sequence</i> → <i>TAB Order</i> → <i>Sequence</i> menus, the <i>Button1</i> to <i>Button4</i> objects are set in the corresponding order. All other objects are removed from the TAB order.</p> <p>The <i>Buttons</i> are acknowledged by pressing the <i>Spacebar</i> or the corresponding hotkey.</p>

C-Action for Setting the Focus

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    Static BOOL bFirst = FALSE;

    //set focus in first run
    if (bFirst==FALSE)
        Set_Focus(lpszPictureName, lpszObjectName);

    BFirst=TRUE;
    Return 100;
}
```

- During the initial call of the function, the focus is set to the own object. The *C-Action* is called once every hour. The focus, however, is set only once.
- This *C-Action* is configured at the *Property* → *Geometry* → *Position X* of the *Option Group1* object in the picture *pictu_5_window_25.pdl*. It is executed in 1 hour cycles.

C-Action for setting the Font Color

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
    long int wValue;

    //get text color
    wValue=GetForeColor(lpszPictureName, "Static Text1");

    //set tag whitch contains text color
    switch (wValue)
    {
        case CO_BLACK: SetTagWord("U16i_pictu_cursor_00",1);
                       break;
        case CO_RED   : SetTagWord("U16i_pictu_cursor_00",2);
                       break;
        case CO_GREEN: SetTagWord("U16i_pictu_cursor_00",4);
                       break;
        case CO_BLUE : SetTagWord("U16i_pictu_cursor_00",8);
                       break;
    }

    //open dialoge box
    SetVisible(lpszPictureName, "Picture Window1", TRUE);
}
}
```

- The property *Font Color* of the object *Static Text1* is set dependent on the value of the *U16i_pictu_cursor_00* tag.
- This *C-Action* is executed after the *OK Button* is pressed in the *pictu_5_window_23.pdl* picture.

C-Action at Open Picture

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPi
{
    //load DLL
    #pragma code ("pdlrtapi.dll")
    #include <pdlrtapi.h>
    #pragma code ()

    PDLRTSetCursorKeys(255,255,255,255,0,0,NULL, (LPVOID)1,NULL);
}
}
```

- If the *pictu_3_chapter_08.pdl* picture is selected, the cursor keys are set by the API function *PDLRTSetCursorKeys*. The first four parameters of the function contain the key codes of the desired keys for moving up/down and left/right.
- In this sample, the WIN key is used for all cursor directions to turn off the cursor option of the keyboard. The runtime cursor can therefore only be moved via the TAB key in the TAB order set.






Note for the General Application


The following adaptations must be made before the general application:

- If multiple windows are used, a key combination for switching among them must be defined in the Control Center. The operating concept in this sample has been designed in such a way that switching among the individual dialogs via the keyboard is not possible and also not required.

- The key combinations and hotkeys used must be adapted to meet your own requirements.
- This example has been designed that no special cursor keys are used to move the runtime cursor but only the TAB key. For the operation of the option groups and check-boxes, however, the cursor keys are used by default.

3.9.2 Cursor Keyboard (example 02)

Valid range	Up to	Font ...
Value Range1	100	
Value Range2	300	
Value Range3	600	
Value Range4	1000	
Other		

This sample is accessed from the *pictu_3_chapter_08.pdl* picture via the key combination *CTRL+W* or the button displayed above using the . It is configured in the *pictu_3_chapter_08a.pdl* picture.

Task Definition

A text is to be entered via the cursor keys and a keyboard configured in a picture. The selection of the individual characters is carried out via the cursor keys. The cursor behavior can be set in runtime via a dialog. This dialog is only displayed after pressing a hotkey.


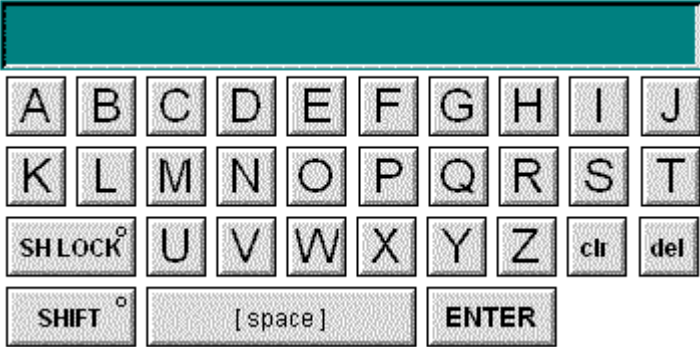


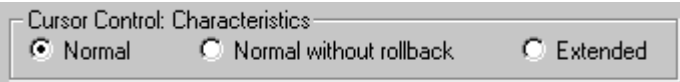
Implementation Concept

For the implementation, a completed keyboard from the library is used. This keyboard can be adapted to meet your own requirements.

For the dialog is displayed in a *Smart Object* → *Picture Window*. To turn on the display of the dialog a *Windows Object* → *Button* is used to which a hotkey has been assigned. The button itself, is not displayed in runtime.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	<p>In Tag Management, two tags of the <i>Unsigned 16-Bit Value</i> type are created that will store the cursor behavior set. In this sample, the <i>U16i_pictu_cursor_04</i> and <i>U16i_pictu_cursor_05</i> tags are used.</p> <p>Additionally, a tag of the <i>Text Tag 16-Bit Character Set</i> type is created to which the entered text is transferred. In the sample, the <i>T16i_pictu_cursor_00</i> tag is used.</p>

Step	Procedure: Implementation in the Graphics Designer
2	<p>Open the Library via the  button on the toolbar.</p> <p>From the Keyboards folder, select the Keyboard Char object and drag it into the picture. In the sample, this is the pictu_3_chapter_08a.pdl picture. The explanation objects can be deleted, the only elements required are displayed below:</p> 
3	<p>For the <i>ENTER</i> Button, create a C-Action at <i>Events</i> → <i>Mouse</i> → <i>Mouse</i> Action that writes the text entered into a text tag. The name of this tag is <i>ConnectedVarChar</i>. Change this name to <i>T16i_pictu_cursor_00</i>.</p> <p>In this sample, the content of the <i>T16i_pictu_cursor_00</i> tag is displayed in a static text having the title of the picture. This is carried out via a tag connection to this tag.</p>
4	<p>Make the settings for the Cursor Control. All objects except for the keys of the keyboard are removed from the TAB order. The TAB order itself needs not to be changed, since the operation of the keyboard is to be performed via the cursor keys and not the TAB key.</p> <p>The settings for the cursor keys of the Cursor Control are made in a C-Action at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i>.</p>
5	<p>The cursor behavior can be set via a dialog box.</p> <p>In normal cases, this is already performed in the <i>Control Center</i>.  on the <i>Computer</i> entry and then select <i>Properties</i> from the pop-up menu. In the following <i>Computer List Properties</i> dialog, click on the  <i>Button Properties</i>. Select the <i>Graphics Runtime</i> tab. In the <i>Cursor Control: Keys</i> field, three settings can be made.</p> 

Step	Procedure: Implementation in the Graphics Designer
6	<p>Create a new picture, which serves as the dialog window. In this sample, this is the <i>pictu_5_window_26.pdl</i> picture.</p> <p>In this picture, configure three <i>Smart Objects</i> → <i>Status Displays</i>. In the sample, these are the <i>Status Display1</i>, <i>Status Display2</i> and <i>Status Display3</i> objects. Via the <i>configuration dialog</i>, set bitmaps for each <i>Status Display</i> that display the pressed status of a button and the not pressed status of a button. The <i>Status 1</i> represents the pressed button and the <i>Status 0</i> the not pressed button.</p> <p>At <i>Properties</i> → <i>State</i> → <i>Current Status</i>, create a <i>Dynamic Dialog</i> each that controls the current status depending on the <i>U16i_pictu_cursor_05</i> tag. This tag contains the temporarily made settings of the cursor behavior.</p> <p>At <i>Events</i> → <i>Keyboard</i> → <i>Press</i>, create a <i>C-Action</i> that writes a value to the <i>U16i_pictu_cursor_05</i> tag representing a certain selection. These values are: <i>0...Normal</i>, <i>1...Normal without Rollback</i>, <i>10...Extended</i></p>
7	<p>In the same picture, configure two <i>Windows Objects</i> → <i>Buttons</i>. In the sample, these are the <i>Button1</i> and <i>Button2</i> objects.</p> <p><i>Button1</i> is used as the <i>OK Button</i>. At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, create a <i>C-Action</i> that writes the value of the <i>U16i_pictu_cursor_05</i> tag to the <i>U16i_pictu_cursor_04</i> tag. This tag represents the current cursor behavior. Afterwards, the API function <i>PDLRTSetCursorKeys</i> switches the cursor behavior. The value stored in the <i>U16i_pictu_cursor_04</i> tag corresponds to a number value expected by the function for a certain cursor behavior. In addition, the focus is set to the <i>A-Button</i> of the keyboard and the window made invisible.</p> <p><i>Button2</i> is used as the <i>Cancel Button</i>. At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, create a <i>C-Action</i> that sets the focus to the <i>A-Button</i> of the keyboard and hides the window.</p>
8	<p>In the <i>pictu_3_chapter_08a.pdl</i> picture, configure a <i>Smart Object</i> → <i>Picture Window</i>, in which the picture just configured is displayed. In this sample, this is the <i>Picture Window1</i> object. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the <i>pictu_3_window_26.pdl</i> picture. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i>.</p>
9	<p>In the <i>pictu_3_chapter_08a.pdl</i> picture, create a <i>Windows Object</i> → <i>Button</i>. In this sample, this is the <i>Button5</i> object.</p> <p>For <i>Button5</i>, create a <i>C-Action</i> that writes the value of the currently set cursor behavior to the <i>U16i_pictu_cursor_05</i> tag and displays the <i>Picture Window1</i> object.</p> <p>In addition, a hotkey is assigned to this <i>Button</i>. In the sample, this is the key combination <i>CTRL+E</i>. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display</i> to <i>No</i>. This hides the button, but leaves the configured hotkey active.</p>

C-Action at the Press Keyboard Event

```
#include "apdefap.h"
void OnKeyDown(char* lpszPictureName, char* lpszObjectName, char* lpszProc
{
if (nChar==VK_SPACE)
    SetTagWord("U16i_pictu_cursor_05", 0);
}
```

- If the runtime cursor is on top of the Status Display, this C-Action will be executed if any key is pressed. The nChar tag contains the key code of the corresponding key. If this is the spacebar, the corresponding value of the cursor behavior is written to the tag. In this sample, this is the value for the normal cursor behavior.
- This C-Action must be configured at the event Press Key, since the object is not a button, but a status display. Otherwise, the event Mouse Action could be used.

To lay out the object for mouse operation, another C-Action must be created at the Mouse Action event for which the query of the key code is omitted.

C-Action at the OK Button

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
//load DLL
#pragma code ("pdlrtapi.dll")
#include <pdlrtapi.h>
#pragma code ()

//set selected cursor mode to tag
SetTagWord("U16i_pictu_cursor_04",
    GetTagWord("U16i_pictu_cursor_05"));

//set cursor mode
PDLRTSetCursorKeys(VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, 0,
    GetTagWord("U16i_pictu_cursor_04"), NULL,
    (LPVOID)1, NULL);

//set focus to A-button
Set_Focus("pictu_3_chapter_08a.PDL", "Button92");

//close window
SetVisible("pictu_3_chapter_08a.PDL", "Picture Window1", FALSE);
}
```

- Loading of the DLL containing the *PDLRTSetCursorKeys* function.
- The cursor behavior selected is stored in the *U16i_pictu_cursor_04* tag.
- Via the API function *PDLRTSetCursorKeys*, the cursor setting is made. The first four parameters of the function contain the key codes of the desired keys for moving up/down and left/right. The sixth parameter is used to transfer the desired cursor behavior to the function. This parameter is already contained - coded correctly - in the *U16i_pictu_cursor_04* tag.
- The focus is reset to the *A-Button* of the keyboard and the dialog is closed.
- At the Open Picture event, the *PDLRTSetCursorKeys* function is called as well and the cursor behavior is set to normal. The DLL has already been loaded at this time. It is not necessary to load it again. However, for the sake of completeness, this is mentioned again.


Note for the General Application

The following adaptations must be made before the general application:

- The key combinations and hotkeys used must be adapted to meet your own requirements.
- The library contains two additional keyboards: a number keyboard and a number/character keyboard. They can be used in the same manner as described in this sample.

3.9.3 Entering Values, Switching Operations (example 03)

Additional examples via CTRL+W >>

This sample is accessed from the *pictu_3_chapter_08a.pdl* picture via the key combination *CTRL+W* or the button displayed above using the . It is configured in the *pictu_3_chapter_08b.pdl* picture.

Task Definition


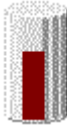
Various control actions are to be performed in a plant picture without using a mouse. Values are to be input and a number of switching operations performed.

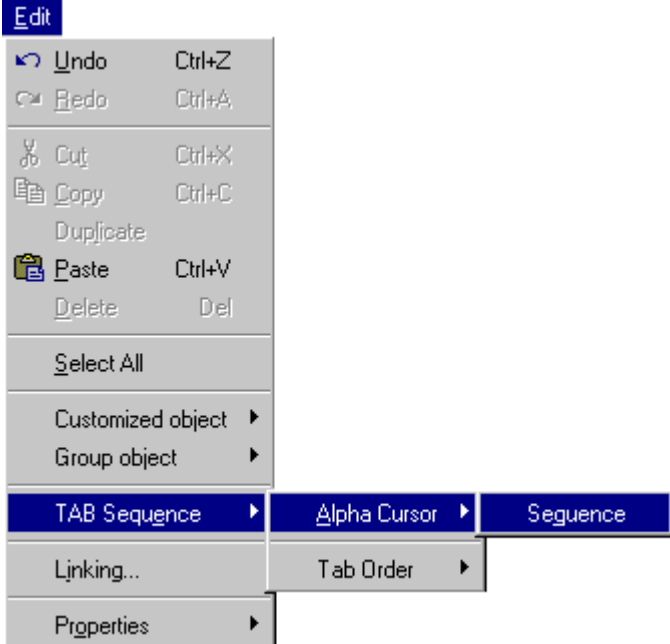


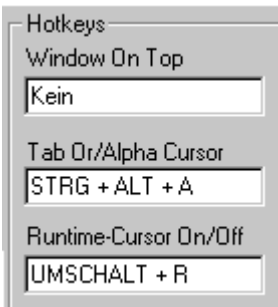
Implementation Concept

For the implementation, *Windows Objects* → *Buttons* are used to which hotkeys are assigned. Values are to be entered in *Smart Objects* → *I/O Fields* and valves turned on and off.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	In Tag Management, create six tags of the <i>Signed 16-Bit Value</i> type that are used to make the entries and to store them afterwards. In the sample, these are the <i>S16i_pictu_cursor_00</i> to <i>S16i_pictu_cursor_05</i> tags.
2	<p>In a picture, configure three <i>Smart Objects</i> → <i>I/O Fields</i> in which the fill level set-point values are to be entered. In this sample, these are the <i>I/O Field1</i>, <i>I/O Field2</i> and <i>I/O Field3</i> objects.</p> <p>For <i>I/O Field1</i>, create a <i>tag connection</i> in the <i>configuration dialog</i> to the <i>S16i_pictu_cursor_00</i> tag and set the <i>High Limit Value</i> to 4999 and the <i>Low Limit Value</i> to 0.</p> <p>Proceed in the same manner for the remaining <i>I/O Field</i>, but set the <i>S16i_pictu_cursor_01</i> or <i>S16i_pictu_cursor_02</i> tags respectively. For <i>I/O Field3</i>, set the <i>Upper Limit Value</i> to 9999.</p> <div style="background-color: #0000ff; color: #ffff00; padding: 5px; text-align: center; width: fit-content; margin: 10px auto;">0000</div>

Step	Procedure: Implementation in the Graphics Designer
3	<p>Configure three <i>Windows Objects</i> → <i>Buttons</i> that are used to apply the values entered in the <i>I/O Fields</i>. In this sample, these are the <i>ButtonF6</i>, <i>ButtonF7</i> and <i>ButtonF8</i> objects.</p> <p>For <i>ButtonF6</i>, create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> that writes the value entered in the <i>S16i_pictu_cursor_00</i> tag to the <i>S16i_pictu_cursor_03</i> tag. Via <i>Properties</i> → <i>Miscellaneous</i> → <i>Hotkey</i>, set the <i>F6</i> hotkey for the <i>Button</i>.</p> <p>Proceed in the same manner for the remaining <i>Buttons</i>.</p> <p>For <i>ButtonF6</i>, create a <i>C-Action</i> at <i>Properties</i> → <i>Geometry</i> → <i>Width</i> that sets the focus to this button.</p> 
4	<p>Configure three <i>Standard Objects</i> → <i>Rectangles</i> that represent the values entered. In this sample, these are the <i>Rectangle9</i>, <i>Rectangle10</i> and <i>Rectangle11</i> objects.</p> <p>Set the <i>Property</i> → <i>Filling</i> → <i>Dynamic Filling</i> to <i>Yes</i>. At <i>Properties</i> → <i>Filling</i> → <i>Fill Level</i>, create a <i>Dynamic Dialog</i> for each rectangle to convert the tag value to a fill level.</p> <p>For the graphical display of the container, several grouped <i>Standard Objects</i> are used.</p> 
5	<p>Configure four additional <i>Windows Objects</i> → <i>Buttons</i>. These buttons are used to turn valves on and off. In this sample, these are the <i>ButtonF9</i>, <i>ButtonF10</i>, <i>ButtonF11</i> and <i>ButtonF12</i> objects.</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, configure a <i>C-Action</i> for each button that inverts a binary tag representing a valve status. Each <i>Button</i> is assigned a hotkey.</p>
6	<p>In the picture, configure four valves that are connected to the corresponding binary tags. A detailed description about creating valves can be found in the chapter <i>Displaying and Hiding Information</i>, in the sample <i>Displaying and Hiding Objects</i> (example 01).</p>

Step	Procedure: Implementation in the Graphics Designer
7	<p>Except for the <i>Buttons</i> to which hotkeys have been assigned, all objects are removed from the TAB order.</p> <p>Via the <i>Edit</i> → <i>TAB Sequence</i> → <i>Alpha Cursor</i> → <i>Sequence</i> menus, the order in which the <i>I/O Fields</i> are selected with the TAB key is defined.</p> 
8	<p>In the <i>Control Center</i>, define a key combination that switches between the TAB Order and the Alpha Cursor.  on the <i>Computer</i> entry and then select <i>Properties</i> from the pop-up menu. In the following <i>Computer List Properties</i> dialog, click on the  <i>Button Properties</i>. Select the <i>Graphics Runtime</i> tab.</p> <p>To switch between the <i>TAB Order/Alpha Cursor</i>, the key combination <i>SHIFT+A</i> is set. Additionally, the key combination <i>SHIFT+R</i> is set to turn the runtime cursor on and off.</p> 

Note:

Via the following button or the *ESC* key, the sample just described can be exited:




Note for the General Application

The following adaptations must be made before the general application:

- The key combinations and hotkeys used must be adapted to meet your own requirements.
- Only the control elements colored red are equipped with functions. All other elements have no function. The entire picture is a schematical representation of the Operator Panel *Simatic OP47*.

3.10 Displaying and Hiding Information

Information

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the . The samples are configured in the *pictu_3_chapter_09.pdl* picture.

3.10.1 Displaying and Hiding Objects (example 01)

In many plant pictures, it sometimes makes sense if certain items of information are not constantly displayed in the picture, but can be shown when required or when specific events occur.




Task Definition

Certain objects or object groups in a picture are to be able to be hidden by the user.

Implementation Concept

To implement this control action, we use a picture in which several valves are displayed. Each valve is assigned a *Windows Object* → *Button* to control the valve, a *Standard Object* → *Static Text* for displaying the name of the valve and a group object representing the status of the valve. In addition, the picture also depicts containers whose fill levels are displayed via *Smart Objects* → *I/O Fields*. Via three *Windows Objects* → *Buttons*, all *I/O Fields*, all *Buttons* and all *Static Texts* can be displayed or hidden.

Implementation in the WinCC Project

Step	Procedure: Displaying and Hiding Objects
1	In Tag Management, create three tags of the <i>Binary Type</i> , which control the visibility of the various object groups. In this sample, the <i>BINi_pictu_info_12</i> , <i>BINi_pictu_info_13</i> , and <i>BINi_pictu_info_14</i> tags are used.
2	In Tag Management, create additional tags of the <i>Binary Tag</i> type that contain the current stati of the valves. The number of tags required depends on the number of valves. In the sample, the <i>BINi_pictu_info_1</i> to <i>BINi_pictu_pictu_11</i> tags are used for a total of 11 valves.
3	To display an open valve, a <i>Standard Object</i> → <i>Polygon</i> is configured, which has the shape of a valve. At <i>Properties</i> → <i>Colors</i> → <i>Background Color</i> , set the color <i>Dark Green</i> . 
4	To display a closed valve, a <i>Standard Object</i> → <i>Polyline</i> is configured, which has the shape of a valve. 
5	Configure two identical <i>Standard Objects</i> → <i>Rectangles</i> and set the background color of the picture at their <i>Properties</i> → <i>Color</i> → <i>Background Color</i> . The <i>rectangles</i> should be slightly larger than the valves, so that they can hide them.
6	Position a <i>rectangle</i> and an open valve one on top of each other and set the open valve to the foreground by clicking on the <i>Button</i>  . Group both objects via the <i>Edit</i> → <i>Group Object</i> → <i>Group</i> menus. For the <i>Group Object</i> generated, configure a <i>tag connection</i> to the <i>BINi_pictu_info_1</i> tag at <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i> .

Step	Procedure: Displaying and Hiding Objects
7	Position the closed valve on top of the second <i>rectangle</i> and set to the foreground. Then position the <i>group object</i> generated at step 6 on top of this and set it to the foreground. Now group these three objects. For the configuration of the remaining valves, this new <i>group object</i> can be copied. Only the <i>tag connection</i> must be adapted.
8	For each valve, configure a <i>Windows Object</i> → <i>Button</i> and create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that negates the corresponding tag value.
9	For each valve, configure a <i>Standard Object</i> → <i>Static Text</i> that contains the name of the valve.
10	Configure several containers whose fill levels are displayed via <i>Smart Objects</i> → <i>I/O Fields</i> .
11	Configure three <i>Windows Objects</i> → <i>Buttons</i> . In this sample, the <i>Button12</i> , <i>Button13</i> and <i>Button14</i> objects are used. For <i>Button12</i> , create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that negates the value of the <i>BINi_pictu_info_12</i> tag. For the remaining <i>Buttons</i> , create <i>C-Actions</i> in the same manner for the <i>BINi_pictu_info_13</i> and <i>BINi_pictu_info_14</i> tags.
12	For all objects that are displayed or hidden via <i>Button12</i> , create a <i>tag connection</i> to the <i>BINi_pictu_info_12</i> tag. Do exactly the same for the other objects. In this sample, <i>Button12</i> makes visible the <i>I/O Fields</i> , <i>Button13</i> the <i>Static Texts</i> and <i>Button14</i> the <i>Buttons</i> .

Note for the General Application

The following adaptations must be made before the general application:

- The basic method of displaying and hiding objects can be adopted.
- The method of displaying the valves can be adopted directly.

3.10.2 Date and Time Display (example 02)

Task Definition

Different ways of displaying the time and date are presented.

Implementation Concept

For the implementation, we will use OCX objects. Additionally, two Standard Objects
 → Static Texts are used, which will display the date and time.

Implementation in the WinCC Project

Step	Procedure: Date and Time Display
1	From the Control selection menu of the object palette, select the <i>WinCC Digital/Analog Clock Control</i> . This generates a time display which you now only have to adjust to the size and type of display you require.
2	Configure a <i>Standard Object</i> → <i>Static Text</i> . In this sample, the <i>Static Text22</i> object is used. At <i>Properties</i> → <i>Font</i> → <i>Text</i> , create a <i>C-Action</i> that reads the current computer time and returns it as the return value. Set the trigger for this action to 1 second.
3	Configure an additional <i>Standard Object</i> → <i>Static Text</i> . In this sample the <i>Static Text23</i> object is used. At <i>Properties</i> → <i>Font</i> → <i>Text</i> , create a <i>C-Action</i> that reads the current date and returns it as the return value.

C-Action for Reading the Time

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    time_t timer;
    struct tm *ptm;
    char *p;

    time(&timer);
    ptm=localtime(&timer);
    p=SysMalloc(9);
    sprintf(p, "%02d:%02d:%02d", ptm->tm_hour, ptm->tm_min, ptm->tm_sec);
    return(p);
}
```

- *time(timer)* returns the current system time in seconds.
- *localtime(timer)* returns a pointer pointing to the system time structure.
- *SysMalloc* reserves a memory area.
- *sprintf* generates a text consisting of a static segment and several numerical segments.

C-Action for Reading the Date

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    time_t timer;
    struct tm *ptm;
    char *p;

    time(&timer);
    ptm=localtime(&timer);
    p=SysMalloc(9);
    sprintf(p, "%02d:%02d:%02d", ptm->tm_mday, ptm->tm_mon, ptm->tm_year);
    return(p);
}
```

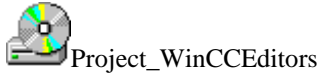
Note for the General Application

The following adaptations must be made before the general application:

- The *WinCC Digital/Analog Clock Control* can be transferred directly to another project.
- The *C-Actions at the Standard Objects* → *Static Texts* can be transferred directly to another project.

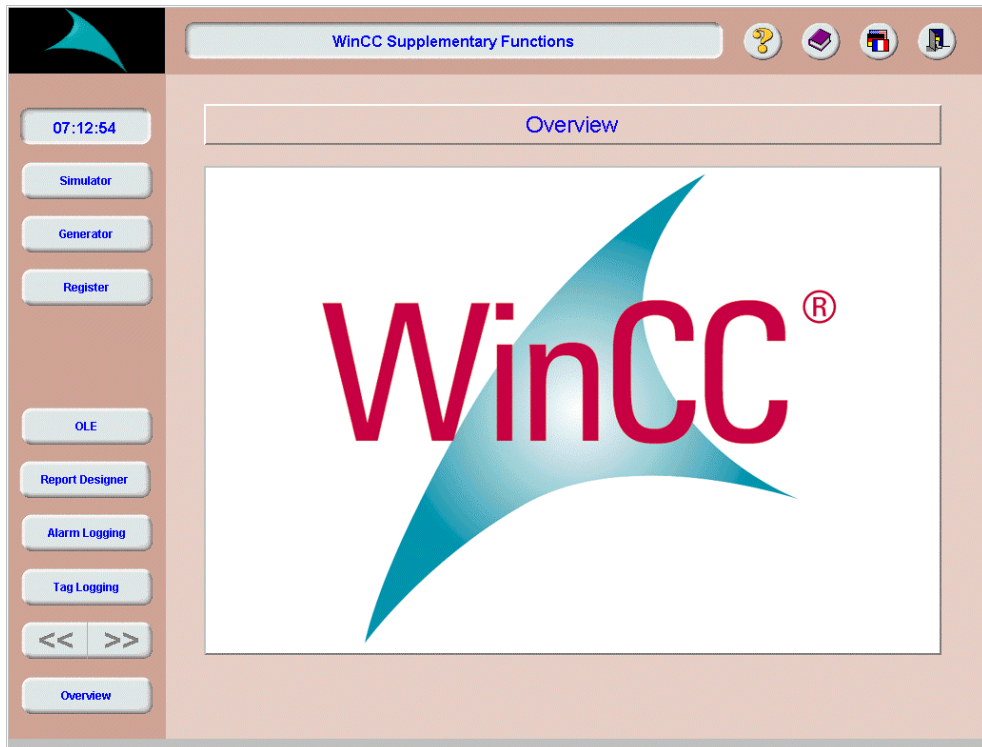
4 WinCC Editors (Project_WinCCEditors)

The project created in this chapter can also be copied directly from the online document to your hard drive. By default, it will be stored to the *C:\Configuration_Manual* folder.




This project presents samples pertaining to the Tag Logging, Alarm Logging and Report Designer editors.

The samples for this topic are configured in the Project_WinCCEditors WinCC project.



4.1 Tag Logging

Tag Logging

In runtime, the samples pertaining to this topic are accessed by selecting the button displayed above using the . The samples are configured in the *ex_3_chapter_01.pdl* to *ex_3_chapter_01f.pdl* pictures.

General Information

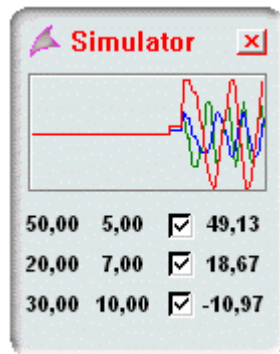
Tag Logging contains functions for applying data from external and internal WinCC tags. This data can be archived using various methods. The display of the data in runtime can be performed in trend or table form.

Simulation of the Process Values

The sample project provides a project-internal simulator for the simulation of process values that are to be archived by *Tag Logging*. This simulator is activated by pressing the corresponding button on the key bar.

Simulator

This simulator makes possible the simulation of three different internal tags with sine wave profiles. Another tag is supplied with the sum of the individual tag values. The tag value profiles are displayed in a small trend window.



Below the trend window, three lines consisting of various input and output elements are located. Each line is assigned to one trend profile. In first I/O field, the amplitude of the trend can be changed. In the second I/O field, the frequency of the trend in oscillations per minute can be set. Via the check-box, the corresponding trend simulation can be stopped. In the final I/O field, the current trend amplitude is displayed. The G64_ex_tlg_01, G64_ex_tlg_02 and G64_ex_tlg_03 tags are supplied with values. The G64i_ex_tlg_04 tag is supplied with the sum of the values of these three tags. If the simulator is deactivated, all tag values are reset to zero.

4.1.1 Cyclic-Continuous Archiving (ex_3_chapter_01.pdl)

Task Definition


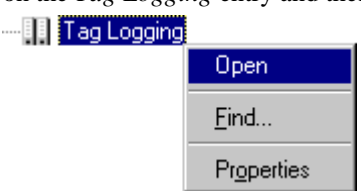


Various process values are to be stored continuously in an archive at a set cycle. The stored data is to be displayed graphically in runtime using trends.

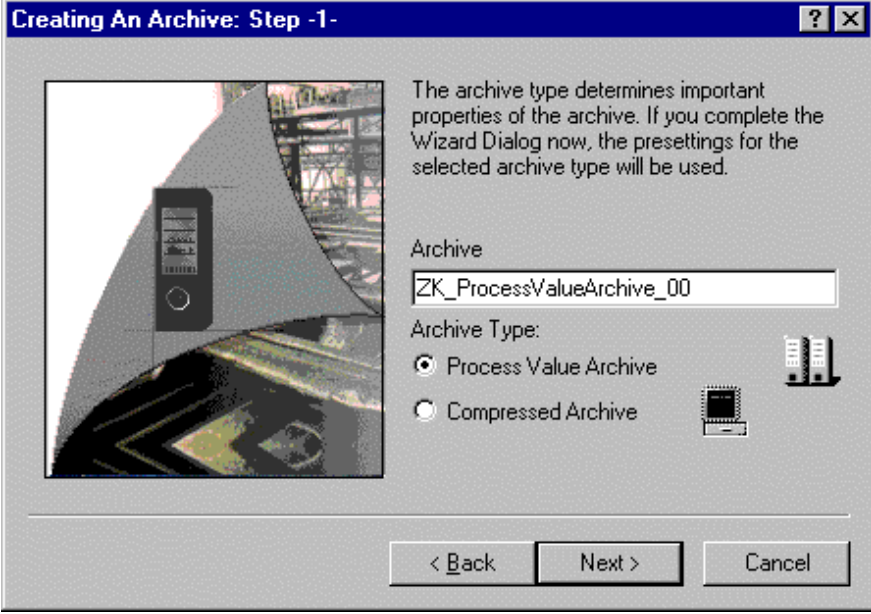
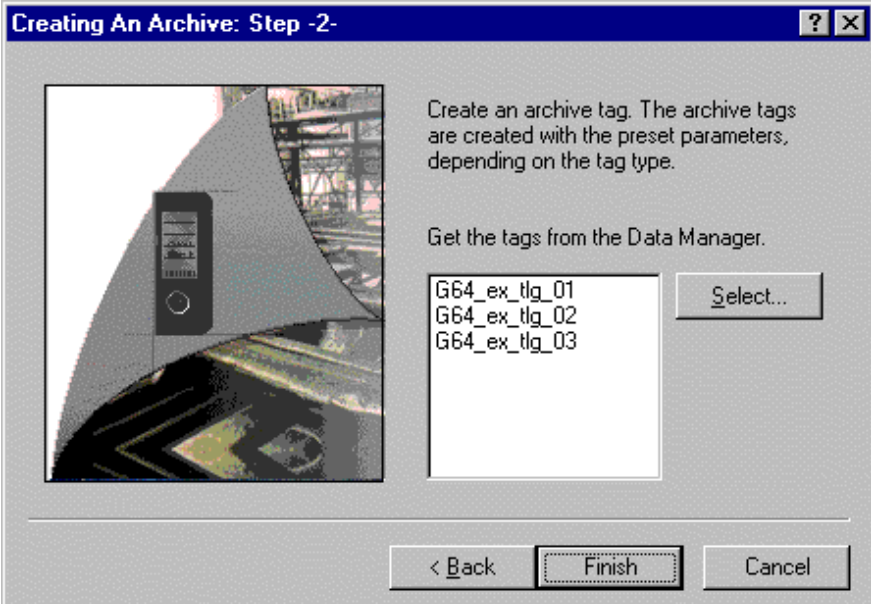
Implementation Concept


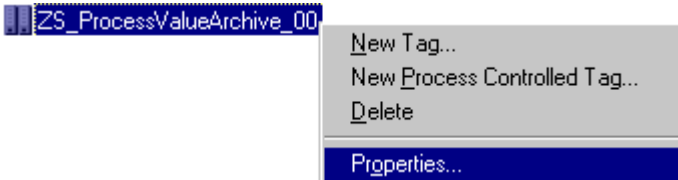
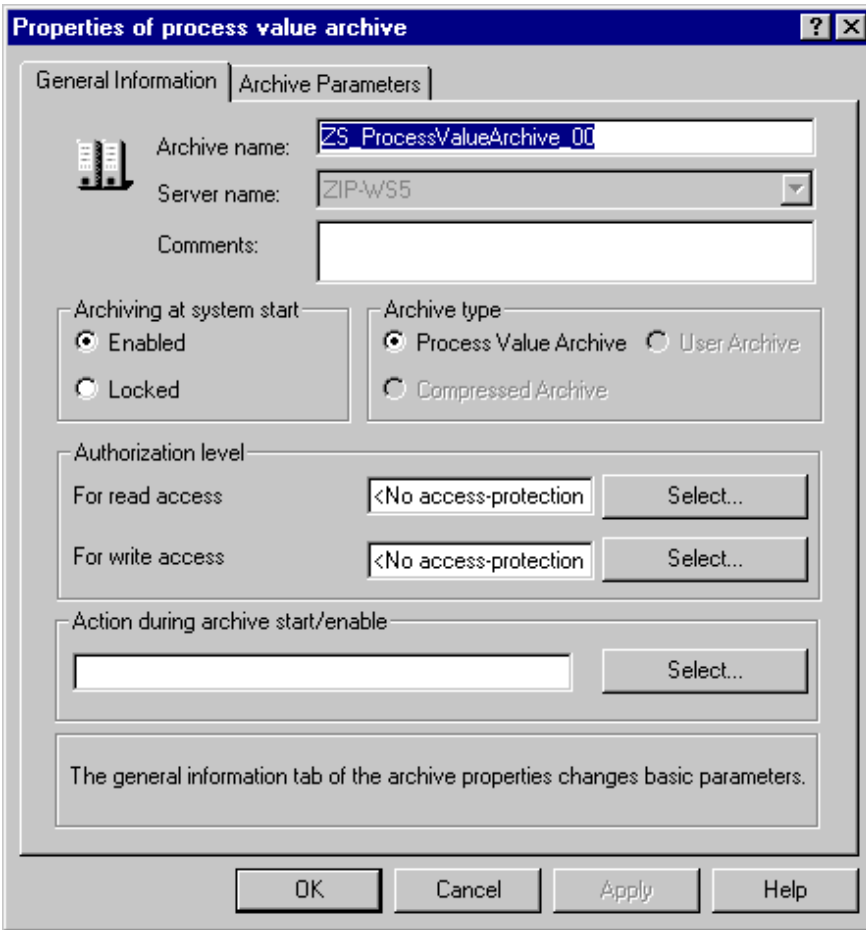
To archive the data to be displayed, a *Cyclic-Continuous Process Value Archive* is created in the *Tag Logging* editor.

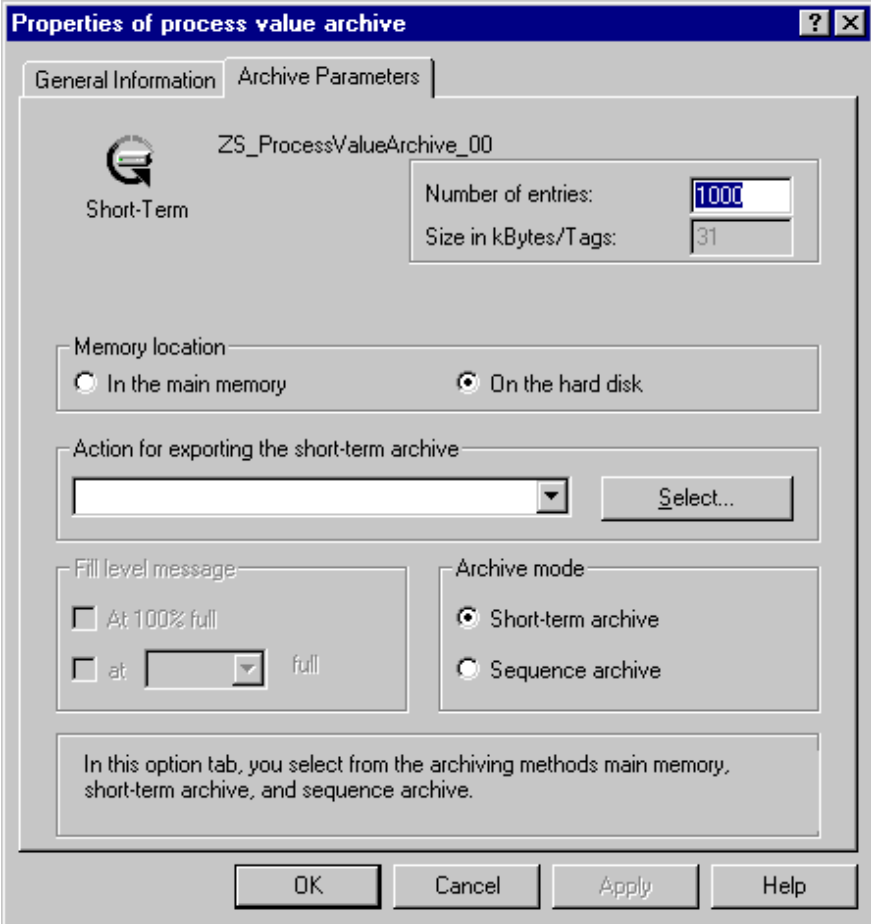

In runtime, the archive is displayed via a special *Control*. This control displays the data in trend form.

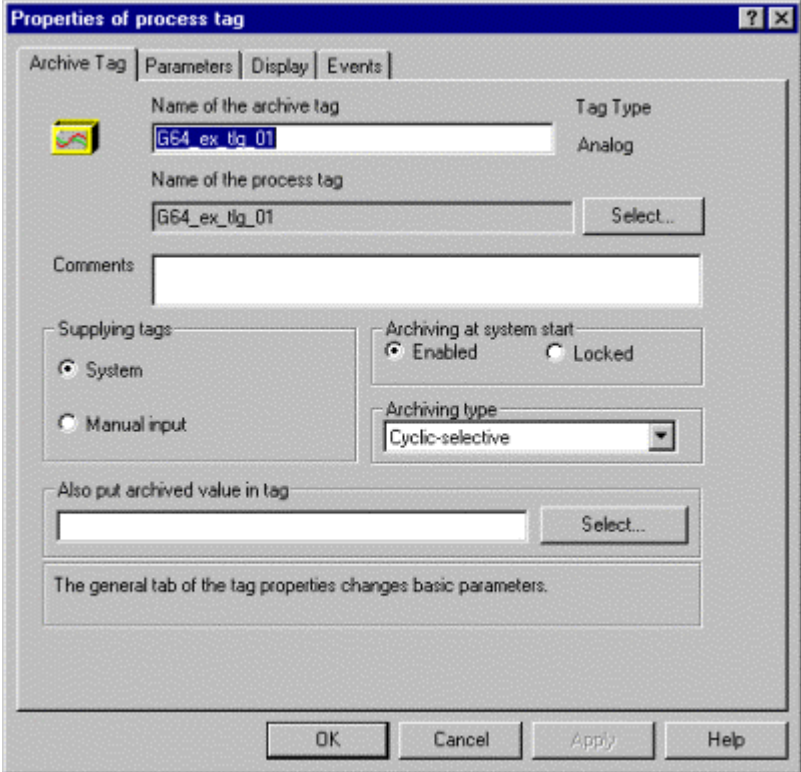
Creating a Process Value Archive

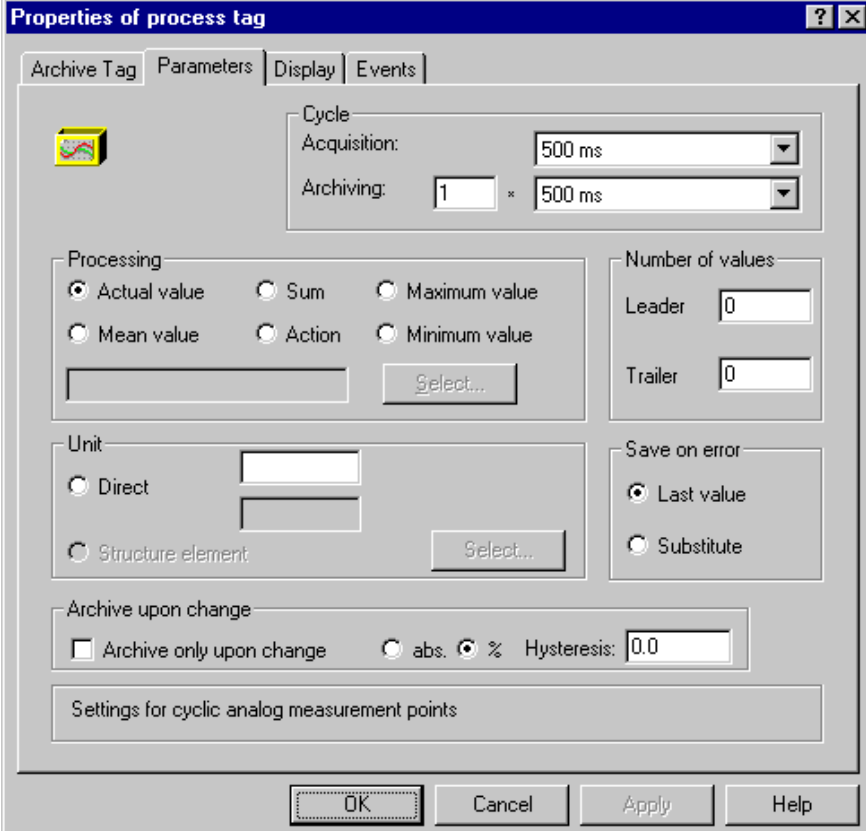
Step	Procedure: Creating a Process Value Archive
1	<p>Create the tags to be archived in Tag Management.</p> <p>In this sample, the <i>G64_ex_tlg_01</i>, <i>G64_ex_tlg_02</i> and <i>G64_ex_tlg_03</i> tags are archived, which are supplied with values by the simulator.</p>
2	<p>Open the <i>Tag Logging</i> editor. This is done from the <i>WinCC Explorer</i> via a  on the <i>Tag Logging</i> entry and then selecting <i>Open</i> from the pop-up menu.</p> 
3	<p>Creating a new archive. Via a  on the <i>Archives</i> entry and then selecting <i>Archive Wizard</i> from the pop-up menu, a Wizard is started. This Wizard guides the user through the creation of a new archive.</p> 

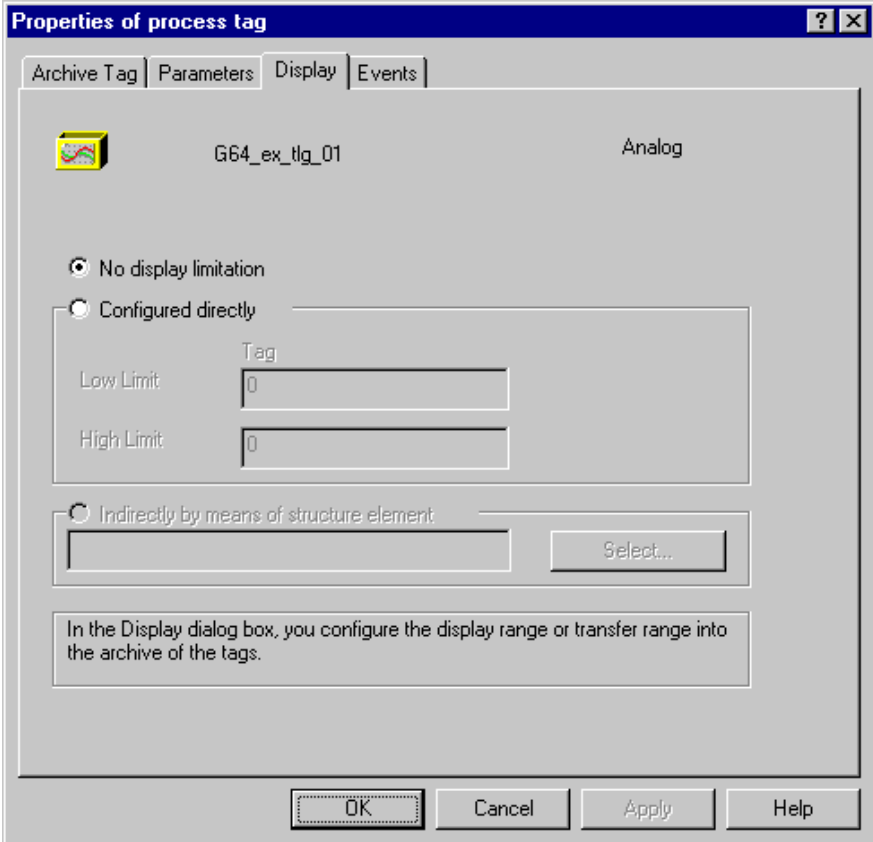

Step	Procedure: Creating a Process Value Archive
4	<p>The start page is exited by clicking on the <i>Next</i> button.</p> <p>On the next page, set as the <i>Archive Type</i> the <i>process value archive</i> option. Enter an <i>Archive Name</i>. In this sample, the archive has been named <i>ZK_ProcessValueArchive_00</i>.</p> <p>Continue to the next page by clicking on <i>Next</i>.</p> 
5	<p>On the third page of the Wizard, the tags to be archived are defined. This is done via the <i>Select</i> button. In this sample, the <i>G64_ex_tlg_01</i>, <i>G64_ex_tlg_02</i> and <i>G64_ex_tlg_03</i> tags are used.</p> <p>Close this page of the Wizard by clicking on <i>Finish</i>.</p> 

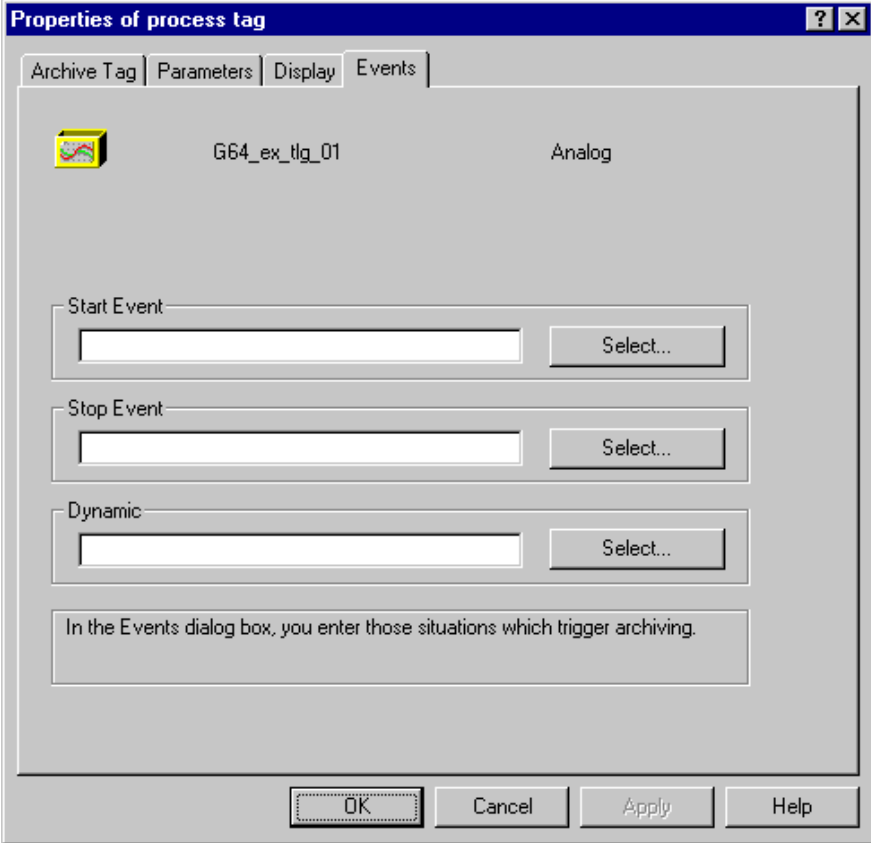
Step	Procedure: Creating a Process Value Archive
6	<p>On the right side of the window, the entry of the newly created archive will be displayed.</p> <p>Via a  on this entry and then selecting <i>Properties</i>, the properties dialog of this archive is opened.</p> 
7	<p>In the <i>General Information</i> tab, the basic archive parameters can be set. The <i>Archive Name</i> and the <i>Archive Type</i> have already been specified in the <i>Archive Wizard</i>. The <i>Archive Type</i> cannot be changed anymore.</p> <p>The <i>Archiving at System Start</i> is enabled.</p> <p>This initiates the archiving directly after the system start. It is not necessary to enable the archive via a separate function. In the <i>Authorization Level</i> field, <i>No Access Protection</i> is set for both the read and write access. The data can be used by any user and are not subject to a special access protection.</p> <p>At the start of the archive, no special action is performed. Such an action could be used, for example, to receive information about the archive status.</p> 

Step	Procedure: Creating a Process Value Archive																
8	<p>In the <i>Archive Parameters</i> tab, additional properties of the archive are set. As the archive size, 1000 data records is set. As the <i>Storage Location</i>, select <i>On the Hard Disk</i>. As the <i>Archive Type</i>, select <i>Short-Term Archive</i>. As the <i>Action for Exporting the Short-Term Archive</i>, a function can be set that is executed automatically if the short-term archive is full. For this sample, no action is specified.</p> <p>With the settings made, 1000 data records will be archived to the hard disk. If the maximum number of data records is exceeded, the oldest archive entry will be deleted and replaced by the new entry.</p> <p>Close the properties dialog of the archive by clicking on <i>OK</i>.</p> 																
9	<p>Specify the properties of the individual archive tags.</p> <p>For this purpose,  on the lower table window and select <i>Properties</i> from the pop-up menu to open the properties dialog of an archive tag.</p> <table border="1" data-bbox="483 1675 1274 1808"> <thead> <tr> <th>Tag name</th> <th>Tag type</th> <th>Comments</th> <th>Last changed</th> </tr> </thead> <tbody> <tr> <td>G64_ex_tlg_01</td> <td>Analog</td> <td>Delete</td> <td>11/12/97</td> </tr> <tr> <td>G64_ex_tlg_02</td> <td>Analog</td> <td></td> <td>11/13/97</td> </tr> <tr> <td>G64_ex_tlg_03</td> <td>Analog</td> <td>Properties...</td> <td>11/13/97</td> </tr> </tbody> </table>	Tag name	Tag type	Comments	Last changed	G64_ex_tlg_01	Analog	Delete	11/12/97	G64_ex_tlg_02	Analog		11/13/97	G64_ex_tlg_03	Analog	Properties...	11/13/97
Tag name	Tag type	Comments	Last changed														
G64_ex_tlg_01	Analog	Delete	11/12/97														
G64_ex_tlg_02	Analog		11/13/97														
G64_ex_tlg_03	Analog	Properties...	11/13/97														

Step	Procedure: Creating a Process Value Archive
10	<p>In the <i>Archive Tag</i> tab, the basic tag properties can be set.</p> <p>The corresponding process tag has already been specified in the <i>im Archive Wizard</i>. As the <i>Name of the Archive Tag</i>, a name can be assigned; in this sample, however, the name of the corresponding process tag is used.</p> <p>In the <i>Supplying Tags</i> field, the <i>System</i> radio-button is selected. In the <i>Archiving at System Start</i> field, the <i>Enabled</i> radio-button is selected. As the <i>Archiving Type</i>, <i>Cyclic-Continuous</i> is set. The settings made mean that the data acquisition is started at the system start and continues in constant time intervals until the system is shut down.</p> <p>The archived value is not written to a tag.</p> 

Step	Procedure: Creating a Process Value Archive
11	<p>In the <i>Parameters</i> tab, additional settings are made.</p> <p>In the <i>Cycle</i> field, set as the <i>Acquisition 500 ms</i> and as the <i>Archiving 1*500 ms</i>. In the <i>Processing</i> field, select the <i>Actual Value</i> radio-button.</p> <p>A <i>Unit</i> is not specified. In the case of an error, the last value is to be saved. The <i>Archiving upon Change</i> check-box is not selected.</p>  <p>The screenshot shows the 'Properties of process tag' dialog box with the 'Parameters' tab selected. The 'Cycle' section has 'Acquisition' set to '500 ms' and 'Archiving' set to '1 * 500 ms'. The 'Processing' section has 'Actual value' selected. The 'Number of values' section has 'Leader' and 'Trailer' both set to '0'. The 'Unit' section has 'Direct' selected. The 'Save on error' section has 'Last value' selected. The 'Archive upon change' section has 'Archive only upon change' unchecked and 'Hysteresis' set to '0.0'.</p>

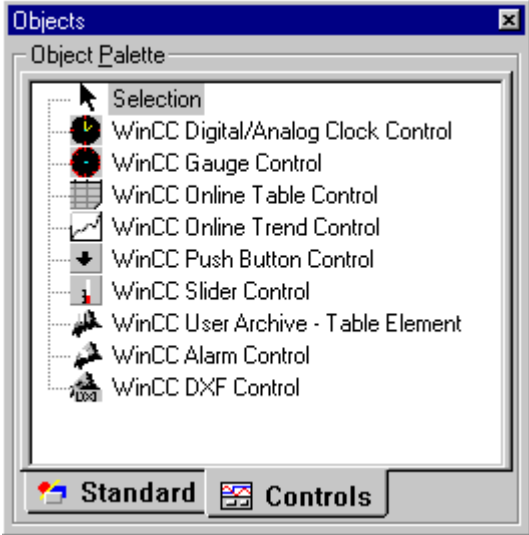
Step	Procedure: Creating a Process Value Archive
12	<p>In the <i>Display</i> tab, the acceptance area of the tag into the archive is specified. In this sample, the <i>No Display Limitation</i> radio-button is selected.</p>  <p>Properties of process tag [?] [X]</p> <p>Archive Tag Parameters Display Events</p> <p> G64_ex_tlg_01 Analog</p> <p><input checked="" type="radio"/> No display limitation</p> <p><input type="radio"/> Configured directly</p> <p>Low Limit: <input type="text" value="0"/> Tag</p> <p>High Limit: <input type="text" value="0"/></p> <p><input type="radio"/> Indirectly by means of structure element</p> <p><input type="text"/> <input type="button" value="Select..."/></p> <p>In the Display dialog box, you configure the display range or transfer range into the archive of the tags.</p> <p><input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Apply"/> <input type="button" value="Help"/></p>

Step	Procedure: Creating a Process Value Archive
13	<p>In the <i>Events</i> tab, no action for changing the archiving cycle is entered for this sample in the <i>Dynamic</i> field.</p> <p>Close the properties dialog of the archive tag by clicking on <i>OK</i>.</p> 
14	<p>The properties of the two other archive tags must also be specified. For this purpose, steps 9 to 13 must be performed.</p>

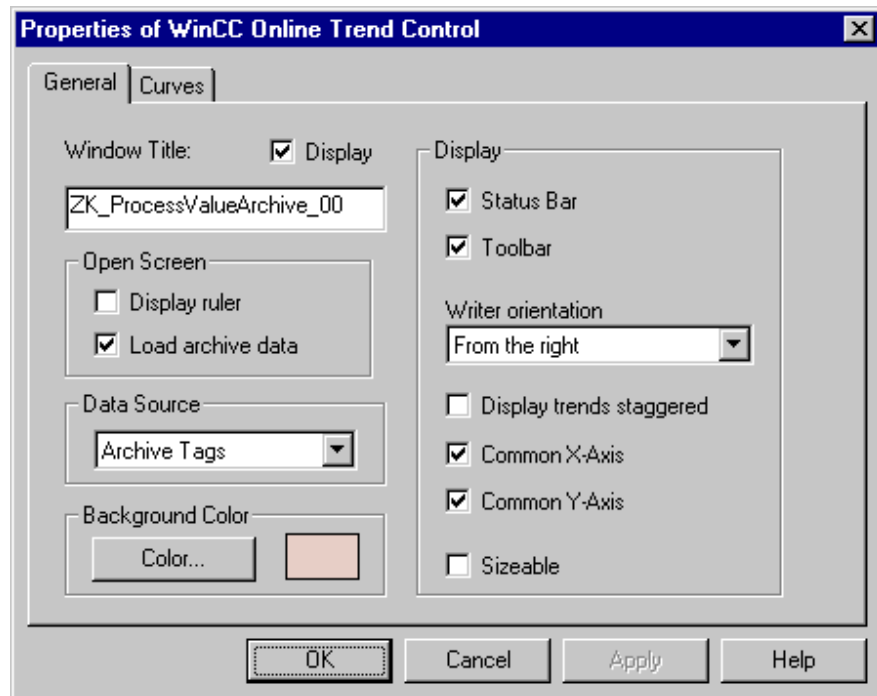
Note:

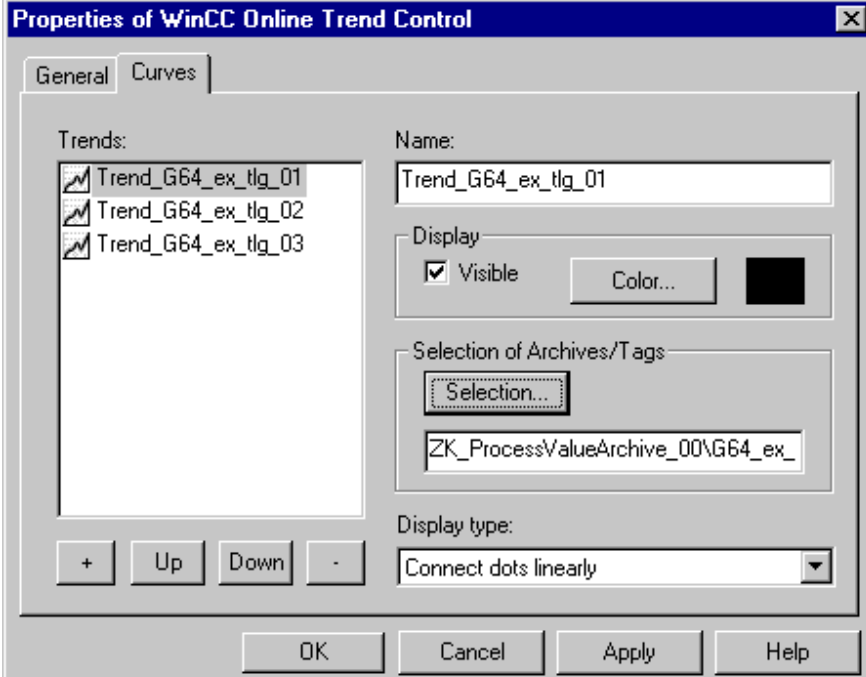
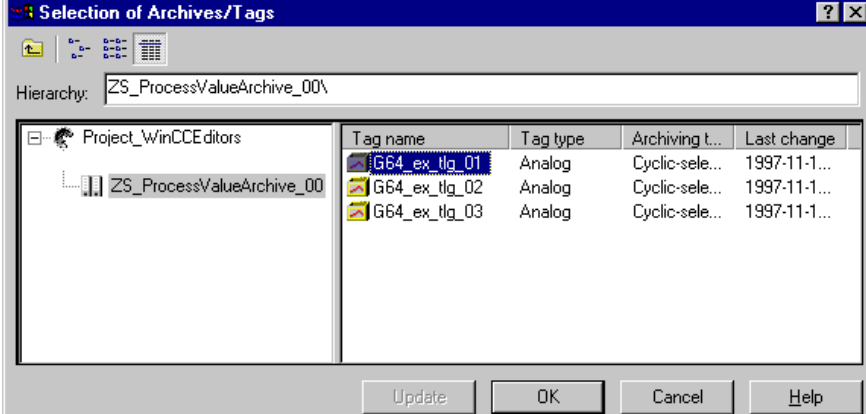
The presettings made by the Archive Wizard during the generation of the process value archive and the corresponding archive tags can be changed by the user via *Archives -> Presettings -> Process Archive* and *Archives -> Presettings -> Analog Tag*. This is advantageous, if a large number of similar archives is to be created.

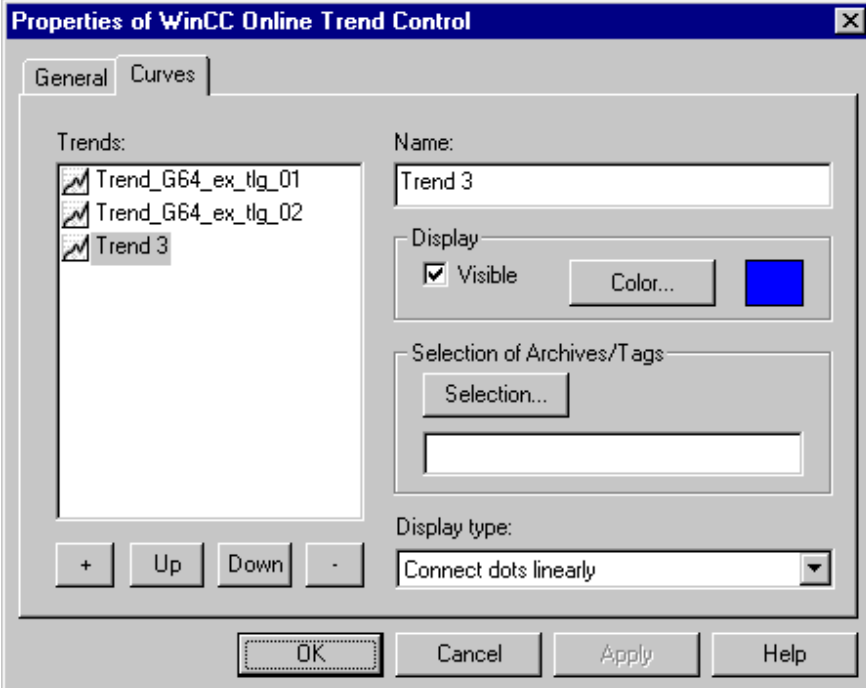

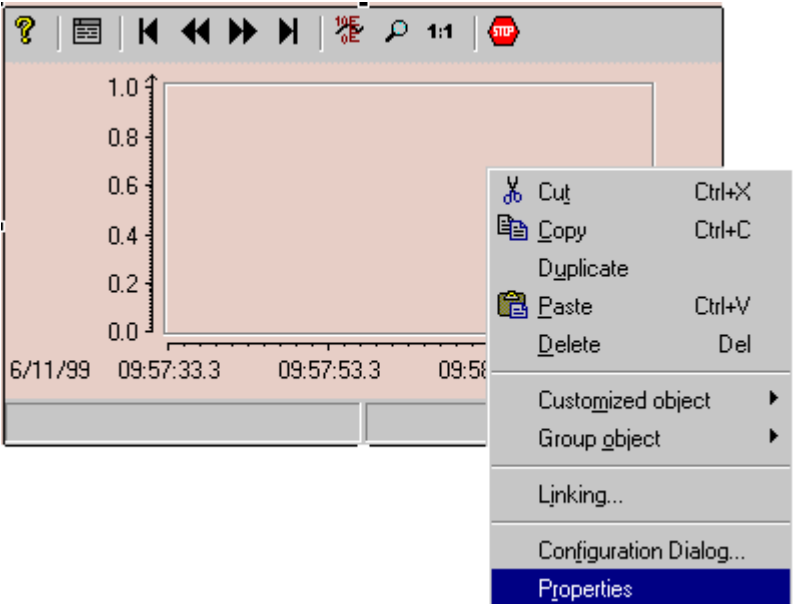
Configuring the Trend Display

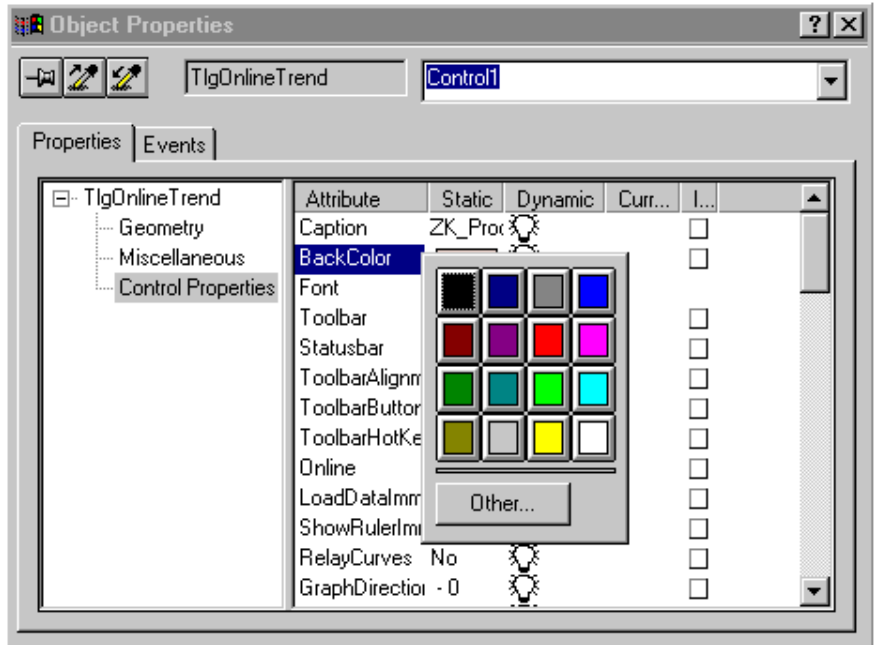
Step	Procedure: Configuring the Trend Display
1	Create a new picture in the <i>Graphics Designer</i> . In this sample, this is the <i>ex_3_chapter_01.pdl</i> picture.
2	<p>Configuration of the Control used for displaying the trend profiles. This is the <i>WinCC Online Trend Control</i>. It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture.</p>  <p>The screenshot shows a window titled 'Objects' with a sub-window 'Object Palette'. The palette is divided into two tabs: 'Standard' and 'Controls'. Under the 'Controls' tab, a list of controls is shown, including 'WinCC Online Trend Control', which is highlighted with a mouse cursor. Other controls listed include 'WinCC Digital/Analog Clock Control', 'WinCC Gauge Control', 'WinCC Online Table Control', 'WinCC Push Button Control', 'WinCC Slider Control', 'WinCC User Archive - Table Element', 'WinCC Alarm Control', and 'WinCC DXF Control'.</p>


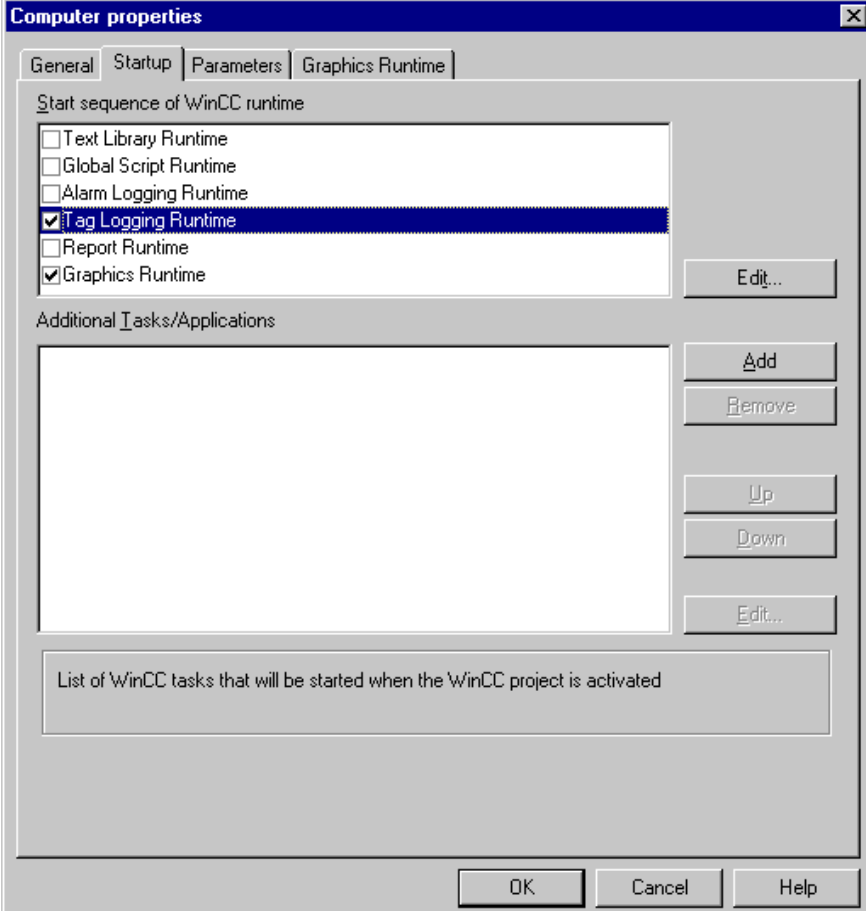
Step	Procedure: Configuring the Trend Display
3	<p>After placing the Control in the picture, its configuration dialog will be opened automatically.</p> <p>In the <i>General Information</i> tab, you can specify the Control's title and how it is labeled. In this sample, the <i>Display</i> check-box is selected and as the <i>Window Title</i> the name of the preciously created archive <i>ZK_ProcessValueArchive_00</i> is entered.</p> <p>In the <i>Open Picture</i> field, you can specify that the ruler window is already displayed when the picture is opened. In this sample, this is not specified. The ruler window must therefore be opened via the corresponding toolbar button if required. Select the <i>Load Data from Archive</i> check-box. If this is not done, the Control will only display values that have been archived after the picture has been opened.</p> <p>In the <i>Supplying Data</i> field, you can select whether the profiles of <i>Archive Tags</i> or <i>Online Tags</i> are to be displayed. If <i>Online Tags</i> is selected, the trend profiles of tags that are not archived can also be displayed. For this sample, <i>Archive Tags</i> is set.</p> <p>Via the <i>Color</i> button, the <i>Background Color</i> of the trend window can be specified. If the entire color palette is to be made available, this must be set as described in step 7 in the <i>Object Properties</i> dialog of the <i>Control1</i> object.</p> <p>In the <i>Display</i> field, this sample specifies the display of the <i>Toolbar</i> and <i>Status Bar</i>. As the <i>Ticker Direction</i>, <i>From the Right</i> is selected. In addition, a <i>Shared X Axis</i> and <i>Shared Y Axis</i> is used. The window size is not to be changed.</p>



Step	Procedure: Configuring the Trend Display																
4	<p>In the <i>Trends</i> tab, the trend profiles to be displayed are specified in detail. One trend has already been created. In this sample, this trend is renamed to <i>Trend_G64_ex_tlg_01</i>. The <i>Display Type Connect Dots Linearly</i> is kept. Via the <i>Selection</i> button, the archive tag to be displayed can be assigned to the trend.</p> 																
5	<p>The <i>Archive/Tag Selection</i> dialog is displayed. In the left window, select the desired <i>ZK_ProcessValueArchive_00</i> archive. In the right window, select the desired <i>G64_ex_tlg_01</i> archive tag available in this archive. Exit the dialog box by clicking on the <i>OK</i> button.</p>  <table border="1" data-bbox="860 1470 1372 1711"> <thead> <tr> <th>Tag name</th> <th>Tag type</th> <th>Archiving t...</th> <th>Last change</th> </tr> </thead> <tbody> <tr> <td>G64_ex_tlg_01</td> <td>Analog</td> <td>Cyclic-sele...</td> <td>1997-11-1...</td> </tr> <tr> <td>G64_ex_tlg_02</td> <td>Analog</td> <td>Cyclic-sele...</td> <td>1997-11-1...</td> </tr> <tr> <td>G64_ex_tlg_03</td> <td>Analog</td> <td>Cyclic-sele...</td> <td>1997-11-1...</td> </tr> </tbody> </table>	Tag name	Tag type	Archiving t...	Last change	G64_ex_tlg_01	Analog	Cyclic-sele...	1997-11-1...	G64_ex_tlg_02	Analog	Cyclic-sele...	1997-11-1...	G64_ex_tlg_03	Analog	Cyclic-sele...	1997-11-1...
Tag name	Tag type	Archiving t...	Last change														
G64_ex_tlg_01	Analog	Cyclic-sele...	1997-11-1...														
G64_ex_tlg_02	Analog	Cyclic-sele...	1997-11-1...														
G64_ex_tlg_03	Analog	Cyclic-sele...	1997-11-1...														




Step	Procedure: Configuring the Trend Display
6	<p>Create two additional trends to display the remaining <i>Archive Tags</i>. A new trend is added in the <i>Trends</i> tab by clicking on the + button. Their properties are set as described in steps 4 to 5. However, the <i>archive tags</i> <i>G64_ex_tlg_02</i> and <i>G64_ex_tlg_03</i> are used. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 
7	<p>Setting the background color of the trend window. For this purpose,  and then select <i>Properties</i> from the pop-up menu to open the <i>Object Properties</i> dialog of the <i>Controll</i> object.</p> 

Step	Procedure: Configuring the Trend Display
	<p>In this sample the <i>BackColor</i> is matched to the color scheme used in the project. You can also make all the settings of the <i>WinCC User Archives Table Control Properties</i> dialog in here. For some settings, however, this is not useful.</p>  <p>The screenshot shows the 'Object Properties' dialog box for the 'TlgOnlineTrend' control. The 'Properties' tab is active, and the 'BackColor' property is selected. A color palette is open, showing a grid of color swatches. The 'Caption' property is set to 'ZK_Proc' and is marked as dynamic. The 'RelayCurves' property is set to 'No' and is also marked as dynamic. The 'GraphDirection' property is set to '-0'. The 'Other...' button is visible at the bottom of the color palette.</p>

Step	Procedure: Configuring the Trend Display
8	<p>Activation of <i>Tag Logging Runtime</i>.</p> <p>For this,  on the <i>Computer</i> entry in the <i>WinCC Explorer</i> and select <i>Properties</i> from the pop-up menu to open the <i>Computer List Properties</i> dialog. Click on the <i>Properties</i> button to open the properties dialog of the local computer.</p> <p>In the <i>Startup</i> tab, the applications to be activated with runtime are selected. The <i>Tag Logging Runtime</i> check-box must be selected.</p> <p>The <i>Computer Properties</i> and <i>Computer List Properties</i> dialogs can be closed by clicking on <i>OK</i>.</p> 

Note regarding the Properties Dialogs:

To set the properties of the *WinCC Online Trend Control*, three different dialogs are available.

- **Configuration Dialog:** This is the dialog that will be opened automatically during the creation of the Control. It provides the user with the most important settings options to allow a quick configuration of the Control. This is the dialog that has been mainly used in the description above. It can be opened via a  on the Control while the SHIFT key is pressed.
- **Properties Dialog of the Control:** This dialog is somewhat more comprehensive. It permits a more exact tuning of the Control to the needs of the user. This dialog can be opened via a  on the Control.
- **Object Properties Dialog:** This is the default properties dialog of the *Graphics Designer*. It can be opened via a  on the Control and then selecting *Properties* from the pop-up menu.

Note for the General Application

The following adaptations must be made before the general application:

- The tags to be archived must be adapted to meet your own requirements.
- This high (fast) archiving cycle selected in this sample makes only sense, if fast changing value profiles are to be displayed. In normal cases, a lower cycle is required. High archiving cycles put high loads on the system.

4.1.2 Cyclic-Selective Archiving (ex_3_chapter_01a.pdl)

Task Definition

Various process values are to be stored continuously in an archive at a set cycle. The archiving is to be started and stopped via a button. The stored data is to be displayed graphically in runtime using trends. A toolbar and status bar with defined objects is to be configured.

Implementation Concept

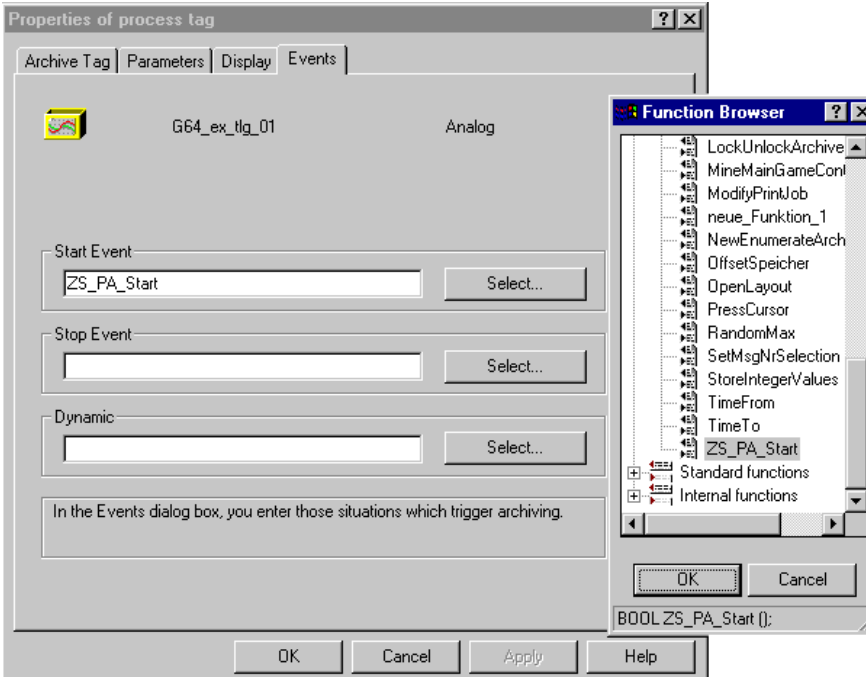
To archive the data to be displayed, a *Cyclic-Selective Process Value Archive* is created in the *Tag Logging* editor.

In runtime, the archive is displayed via a special *Control*. This control displays the data in trend form. The required toolbar is implemented using various *Buttons*, *Status Displays* and *Graphic Objects*. The status bar is implemented using two *Buttons*.

To control the archive, a *project function* is needed that starts and stops the archiving.

Creating a Process Value Archive

Step	Procedure: Creating a Process Value Archive
1	<p>Create the tags to be archived in Tag Management.</p> <p>In this sample, the <i>G64_ex_tlg_01</i>, <i>G64_ex_tlg_02</i> and <i>G64_ex_tlg_03</i> tags are archived, which are supplied with values by the simulator.</p> <p>Create an additional tag of the <i>Binary Tag</i> type, which will store the current status of the archive. In this sample, this is the <i>BINi_ex_tlg_00</i> tag.</p>
2	<p>Creation of a <i>project function</i> in the <i>Global Script</i> editor to start and stop the archiving.</p> <p>In this sample, this is the <i>ZS_PA_Start</i> function. Its functionality is described following this table.</p>
3	<p>Creation of a <i>Process Value Archive</i> in the <i>Tag Logging</i> editor.</p> <p>This is done via the <i>Archive Wizard</i>. In this sample, the archive has been named <i>ZS_ProcessValueArchive_00</i>. For the archiving, the <i>G64_ex_tlg_01</i>, <i>G64_ex_tlg_02</i> and <i>G64_ex_tlg_03</i> tags are selected.</p>
4	<p>Setting the properties of the <i>Process Value Archive</i>.</p> <p>The size of the archive is set to <i>1000</i> data records in the <i>Archive Parameters</i> tab. For the remaining options, the default settings are kept.</p>

Step	Procedure: Creating a Process Value Archive
5	<p>Setting the properties of the <i>Archive Tags</i>.</p> <p>For each of the three tags, <i>cyclic-selective</i> is selected as the <i>Archiving Type</i> in the <i>Archive Tag</i> tab.</p> <p>This type of archiving gives you the option to set a <i>Start Event</i> and a <i>Stop Event</i> in the <i>Events</i> tab. In this sample, the previously created <i>project function ZS_PA_Start</i> is set as the <i>Start Event</i>.</p> <p>For the remaining options, the default settings are kept.</p> 

Project Function ZS_PA_Start

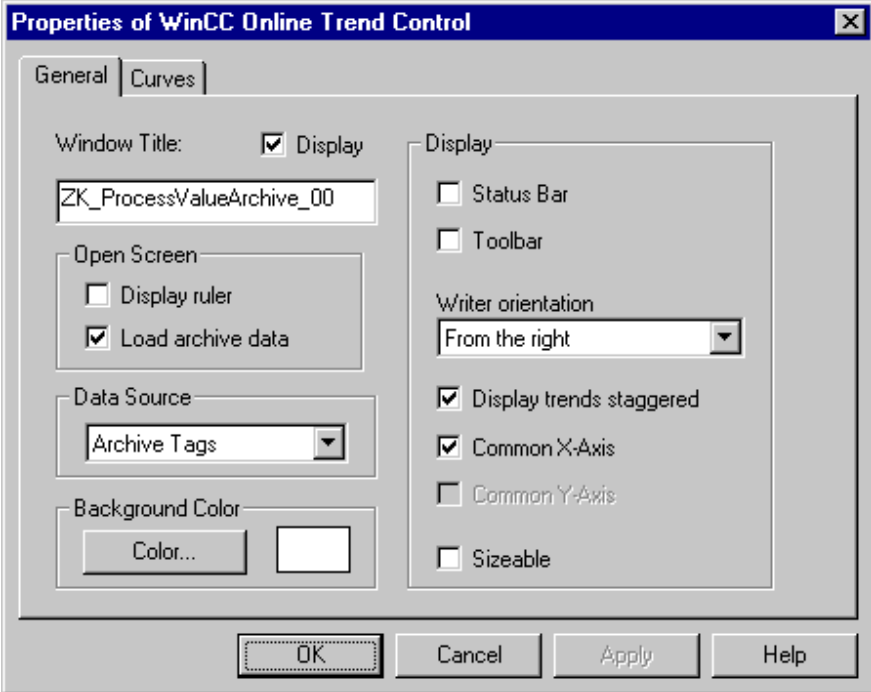
```

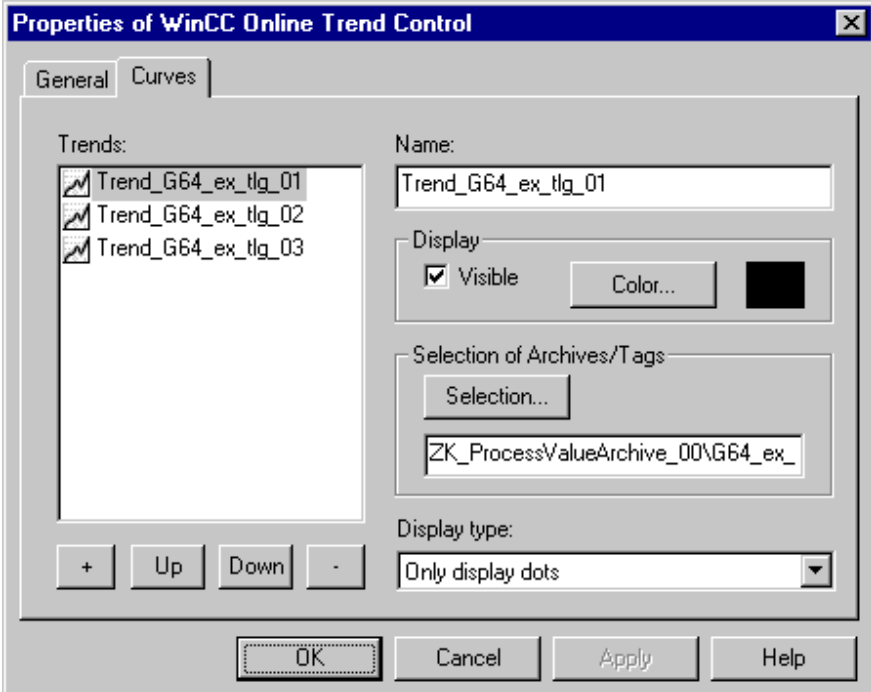
BOOL ZS_PA_Start()
{
    if (GetTagBit("BINi_ex_tlg_00"))
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}



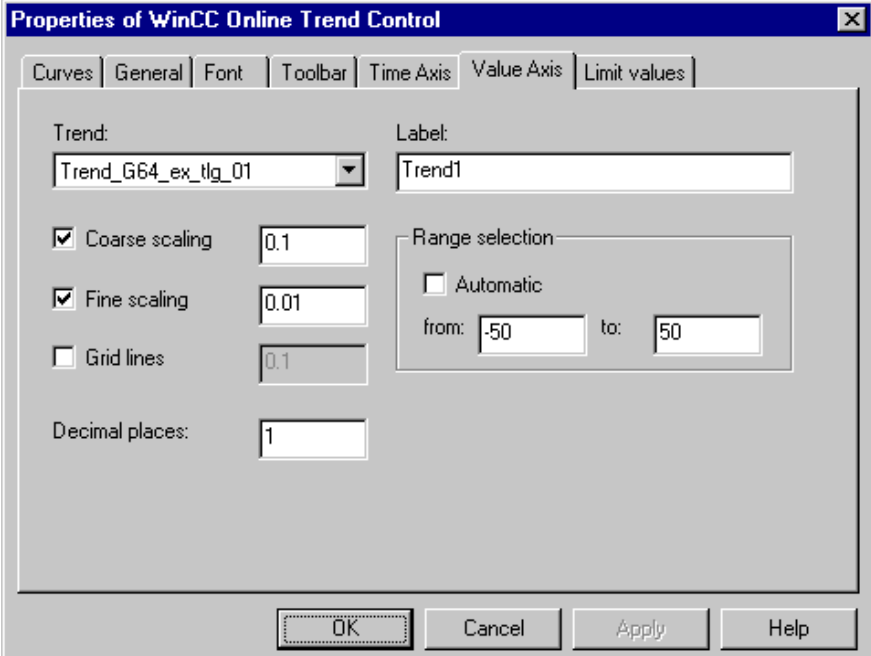
```

- This function returns, depending on the status of the binary tag *BINi_ex_tlg_00*, the value *TRUE* or *FALSE*.
- This function is called by *Tag Logging* at every archiving cycle. Via the return value, the decision is made whether archiving is performed or not. The return value *TRUE* starts the archiving.

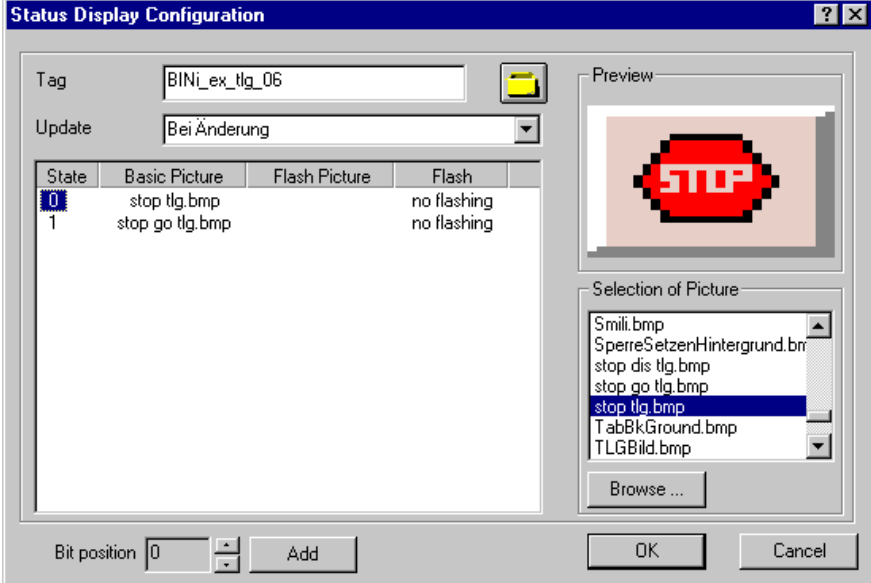
Configuring the Trend Display




Step	Procedure: Configuring the Trend Display
1	Create a new picture in the <i>Graphics Designer</i> . In this sample, this is the <i>ex_3_chapter_01a.pdl</i> picture.
2	<p>Configuration of the Control used for displaying the trend profiles. This is the <i>WinCC Online Trend Control</i>. It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture. After placing the Control in the picture, its configuration dialog will be opened automatically.</p> <p>In the <i>General Information</i> tab, you can specify the Control's title and how it is labeled. In this sample, the <i>Display</i> check-box is deselected. A <i>Window Title</i> is still entered. In the <i>C-Actions</i> created later, this window title is used to reference the corresponding Control. The name of the previously created archive <i>ZS_ProcessValueArchive_00</i> is used.</p> <p>Via the <i>Color</i> button, the <i>Background Color</i> of the trend window is set to white.</p> <p>In the <i>Display</i> field, this sample specifies that the <i>Toolbar</i> and <i>Status Bar</i> are not displayed. The <i>Stagger Trends</i> check-box is selected. This means, that every trend is displayed using a separate diagram.</p> <p>For the remaining options, the default settings are kept.</p> 

Step	Procedure: Configuring the Trend Display
3	<p>In the <i>Trends</i> tab, the trend profiles to be displayed are specified in detail. Three trends are created. The <i>G64_ex_tlg_01</i> to <i>G64_ex_tlg_03</i> tags of the <i>ZS_ProcessValueArchive_00</i> archive are assigned to these trends. The color of the three trends is set to black and the <i>Display Type</i> is set to <i>Show only Dots</i>.</p> <p>For the remaining options, the default settings are kept. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 

Step	Procedure: Configuring the Trend Display
4	<p>Specific properties settings of the individual trends. For this purpose, an expanded properties dialog is available. This dialog is opened via a  on the Control. The properties dialog described previously, on the other hand, is opened via a  on the Control while the CTRL key is pressed.</p> <p>The expanded properties dialog contains in addition to the already mentioned <i>General Information</i> and <i>Trends</i> tabs five additional tabs. In this sample, settings are made only in the <i>Value Axis</i> tab.</p> <p>In the <i>Trend</i> field, set the entry <i>Trend_G64_ex_tlg_01</i> to define the properties of this trend. In the <i>Label</i> field, the text <i>Trend1</i> is entered. The <i>Range Selection</i> is not be performed automatically, but is set from <i>-50</i> to <i>50</i>. For the remaining options, the default settings are kept.</p> <p>The properties of the remaining trends are set in the same manner as just outlined. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 

Configuration of the Toolbar and Status Bar

Step	Procedure: Configuration of the Toolbar and Status Bar
1	In Tag Management, create an internal tag of the <i>Binary Tag</i> type. In this sample, this is the <i>BINi_ex_tlg_06</i> tag.
2	<p>To control the update, a <i>Smart Object</i> → <i>Status Display</i> is configured. In this sample, the <i>Status Display5</i> object is used.</p> <p>Via its <i>configuration dialog</i>, the object is connected to the <i>BINi_ex_tlg_06</i> tag and triggered upon change. The stati <i>0</i> and <i>1</i> are created. In this sample, the bitmaps <i>stop tlg.bmp</i> and <i>stop go tlg.bmp</i> are assigned to these stati.</p> <p>The object's <i>configuration dialog</i> can be exited by clicking on <i>OK</i>.</p> 
3	<p>For the just configured <i>Status Display5</i> object, create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>.</p> <p>This <i>C-Action</i> simulates the pressing of the Stop/Go button of the Control's standard toolbar. In addition, the status of the <i>BINi_ex_tlg_06</i> tag is inverted to display the changed status of the Control's update. A tag value of zero corresponds to an activated update.</p> <p>The status of the <i>BINi_ex_tlg_06</i> tag is always zero at the opening of the picture, since the update of the trend window is always activated when the picture is opened. This is implemented via a <i>direct connection</i> at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> of the Picture Object <i>ex_3_chapter_01a.pdl</i>. This sets the status of the tag to <i>0</i>.</p>

Step	Procedure: Configuring the Trend Display
4	<p>As described in step 2, configure a second <i>Smart Object</i> → <i>Status Display</i>. In this sample, this is the <i>Status Display6</i> object. This object is used to control the archiving.</p> <p>This object is connected to the <i>BINi_ex_tlg_00</i> tag created in the previous section. Correspondingly, different bitmaps are used (<i>Archive.bmp</i> / <i>Archive inv.bmp</i>).</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, a <i>C-Action</i> is created. This action inverts the <i>BINi_ex_tlg_00</i> tag. This tag is used to display the changed status of the archiving and to forward this information to the archive via the <i>project function ZS_PA_Start</i>.</p> 
5	<p>In order to navigate in the archive while the update is stopped, replicas of the four navigation buttons of the Control's standard toolbar are needed.</p> <p>For the implementation, four <i>Windows Objects</i> → <i>Buttons</i> are configured; in this sample, these are the <i>Button4</i>, <i>Button7</i>, <i>Button8</i> and <i>Button11</i> objects.</p> <p>For each of these objects, a <i>C-Action</i> is created at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>. These actions simulate the pressing of the buttons on the standard toolbar.</p> <p>in addition, a <i>Smart Object</i> → <i>Graphic Object</i> is required that places itself over these buttons and makes them inoperational in case the update is started. In this sample, this is the <i>Graphic Object2</i>. This object displays the four buttons in an inoperational status (<i>Pfeile dis.bmp</i>). At <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i>, create a <i>Dynamic Dialog</i>. This dialog controls the visibility of the object dependent on the <i>BINi_ex_tlg_06</i> tag, which contains information about the update of the Control.</p> 
6	<p>To display the status bar, two <i>Windows Objects</i> → <i>Buttons</i> are configured; in this sample, these are the <i>Button5</i> and <i>Button6</i> objects.</p> <p>For the text display, <i>Buttons</i> are used since they can easily be equipped with a 3D border. For this purpose, therefore, no additional objects are required.</p> <p>For <i>Button5</i>, create a <i>C-Action</i> at <i>Properties</i> → <i>Font</i> → <i>Text</i>. This action either returns the text <i>Archiving Started</i> or <i>Archiving Stopped</i> to the property dependent on the <i>BINi_ex_tlg_00</i> tag. A <i>C-Action</i> instead of an equally applicable <i>Dynamic Dialog</i> is used to realize a language switch.</p> <p>Proceed in the same manner as just described for <i>Button6</i> with the <i>BINi_ex_tlg_06</i> tag.</p> 

Note:

The implementation of the time selection and print preview buttons is detailed in the *Report Designer* chapter, *Printing Trend Windows in Runtime* (ex_3_chapter_01a.pdl) example.

C-Action at the Stop/Go Button Object (Status Display5)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszI
{
TlgTrendWindowPressStartStopButton("ZS_ProcessValueArchive_00");
SetTagBit("BINi_ex_tlg_06", (SHORT)!GetTagBit("BINi_ex_tlg_06"));
}
```

- The call of the standard function *TlgTrendWindowPressStartStopButton* has the same effect as pressing the *Stop/Go Button* on the Control's standard toolbar. A text is assigned to the function to allow it to identify the Control to be accessed. This text is the window title that has been specified during the configuration of the Control. In this sample, this was the text *ZS_ProcessValueArchive_00*.
- Inverting the *BINi_ex_tlg_06* tag to store the current status of the Control's update.

C-Action at the Navigation Button Start (Button4)

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper!
{
TlgTrendWindowPressFirstButton("ZS_ProcessValueArchive_00");
}
```

- The call of this standard function has the same effect as pressing the *First Data Record* button on the Control's standard toolbar. The functions used at the other buttons are:
 - *TlgTrendWindowPressPrevButton*
 - *TlgTrendWindowPressNextButton*
 - *TlgTrendWindowPressLastButton*

Note:

For each button on the standard toolbar of the *WinCC Online Trend Control*, a corresponding standard function is available which simulates the pressing of each button.

C-Action for Displaying the Status Bar Text (Button5)

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
char start[40] = "";
char stop [40] = "";

switch(GetLanguage())
{
case 1031 : strcpy(start, "  Archivierung gestartet... ");
           strcpy(stop, "  Archivierung gestoppt... ");
           break;
case 1033 : strcpy(start, "  Archiving started... ");
           strcpy(stop, "  Archiving stoped... ");
           break;
case 1036 : strcpy(start, "  Archivage démarré...");
           strcpy(stop, "  Archivage arrêté...");
           break;
default   : strcpy(start, "  Archivierung gestartet... ");
           strcpy(stop, "  Archivierung gestoppt... ");
           break;
}

if (GetTagBit("BINi_ex_tlg_00"))
{
return start;
}
else
{
return stop;
}
}
```

- Creation of two text tags. In these tags, dependent on the currently set language, a text for the started and stopped archiving status is entered. The currently set language is determined via the *GetLanguage()* function.
- Dependent on the *BINi_ex_tlg_00* tag, the text in the *start* tag or *stop* tag is returned to the property. The action is triggered upon the change of the *BINi_ex_tlg_00* tag.

Note:

In the following sample, a status bar with separate objects is implemented as well. For illustrative purposes however, a *Dynamic Dialog* instead of a *C-Action* is used to control the status bar.

Note for the General Application

The following adaptations must be made before the general application:

- The tags to be archived must be adapted to meet your own requirements.
- This high (fast) archiving cycle selected in this sample makes only sense, if fast changing value profiles are to be displayed. In normal cases, a lower cycle is required. High archiving cycles put high loads on the system.
- The archiving start and archiving end can be made dependent upon certain events - it does not have to be the pressing of a button.
- The appearance of the required elements can be adapted to meet your own needs. The same applies to the status bar.
- The display type has been selected to better show time intervals in which no archiving took place. In all other display types, all points of the archive are linked. This would mean that intervals in which no archiving took place are bridged by a line.

4.1.3 Archiving if Values are Exceeded (ex_3_chapter_01b.pdf)

Task Definition

A process value is to be stored in the archive one time at the moment a certain limit value is exceeded. The stored values are to be displayed in a table. The chronological progress of this process value is to be displayed as a trend. A toolbar and status bar with defined objects is to be configured.

Implementation Concept

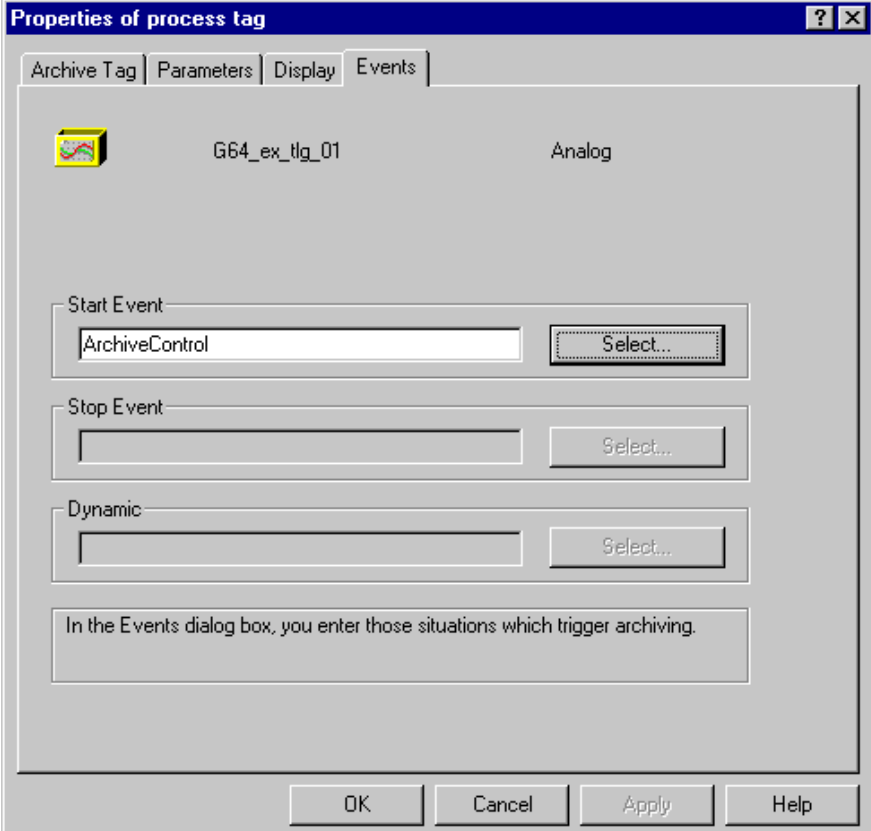

To archive the data to be displayed, an *Acyclic Process Value Archive* is created in the *Tag Logging* editor.

In runtime, the archive is displayed via a special *Control*. This Control displays the data in table form. The trend profile of the process value is displayed in another Control. The required toolbar is implemented using various *Buttons*, *Status Displays* and *Graphic Objects*. The status bar is implemented using a *Button*.

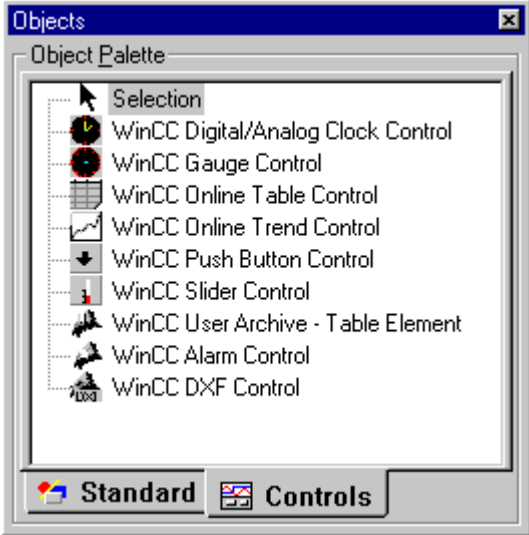
To control the archive, a *project function* is created. This function triggers the archiving, if the process value exceeds a certain limit value.

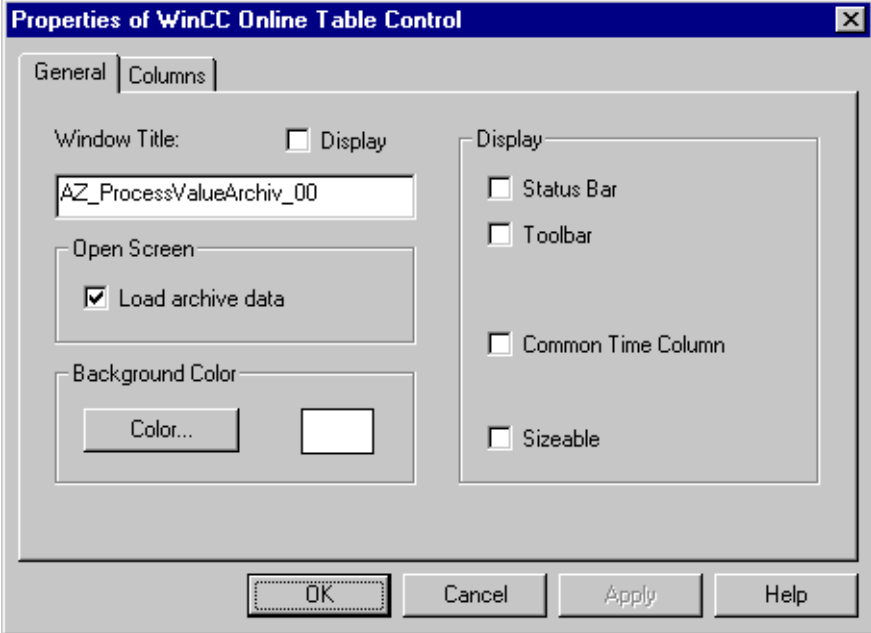
Creating a Process Value Archive

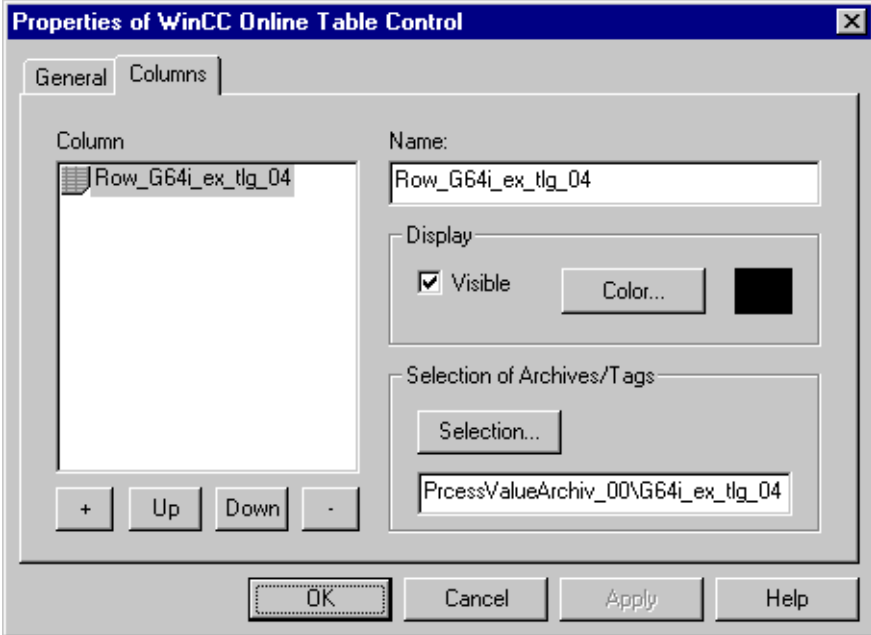
Step	Procedure: Creating a Process Value Archive
1	In <i>Tag Management</i> , create two tags. One tag is supplied with the sum of the values provided by the simulator. In this sample, this is the <i>G64I_ext_lgl_04</i> tag. The other tag is archived, if a limit value is exceeded. In this sample, this is the <i>G64i_ex_tlg_08</i> tag.
3	Creation of a <i>Process Value Archive</i> in the <i>Tag Logging</i> editor. This is done via the <i>Archive Wizard</i> . In this sample, the archive has been named <i>AZ_ProcessValueArchive_00</i> . For the archiving, the <i>G64i_ex_tlg_08</i> tag is selected.
4	Setting the properties of the <i>Process Value Archive</i> . The size of the archive is set to 25 data records in the <i>Archive Parameters</i> tab. For the remaining options, the default settings are kept.



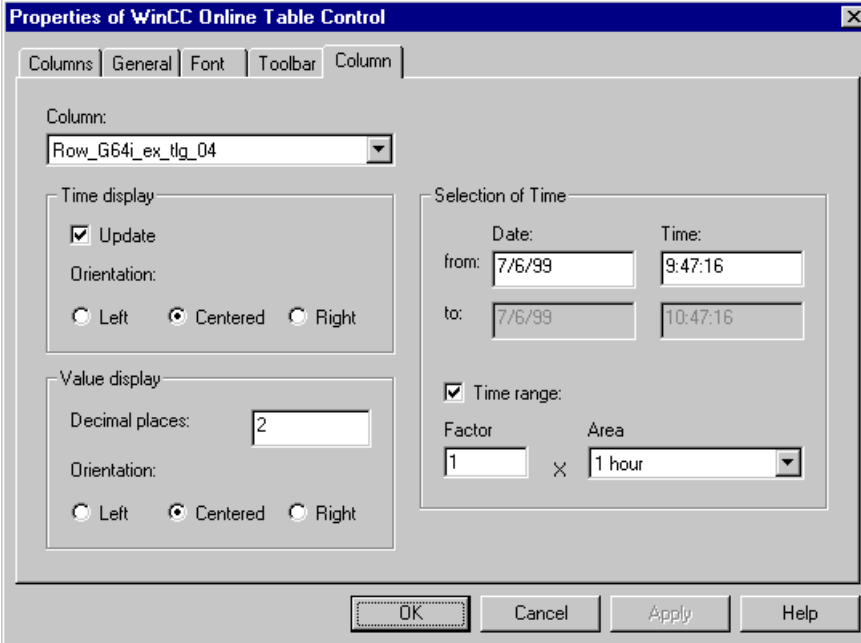

Step	Procedure: Creating a Process Value Archive
5	<p>Setting the properties of the <i>Archive Tags</i>.</p> <p>In the <i>Archive Tag</i> tab, <i>acyclic</i> is selected as the <i>Archiving Type</i>. This type of archiving archives every time the archive tag changes. For the remaining options, the default settings are kept.</p>  <p>Properties of process tag [?] [X]</p> <p>Archive Tag Parameters Display Events</p> <p> G64_ex_tlg_01 Analog</p> <p>Start Event ArchiveControl [Select...]</p> <p>Stop Event [Select...]</p> <p>Dynamic [Select...]</p> <p>In the Events dialog box, you enter those situations which trigger archiving.</p> <p>[OK] [Cancel] [Apply] [Help]</p>

Configuration of the Table Display

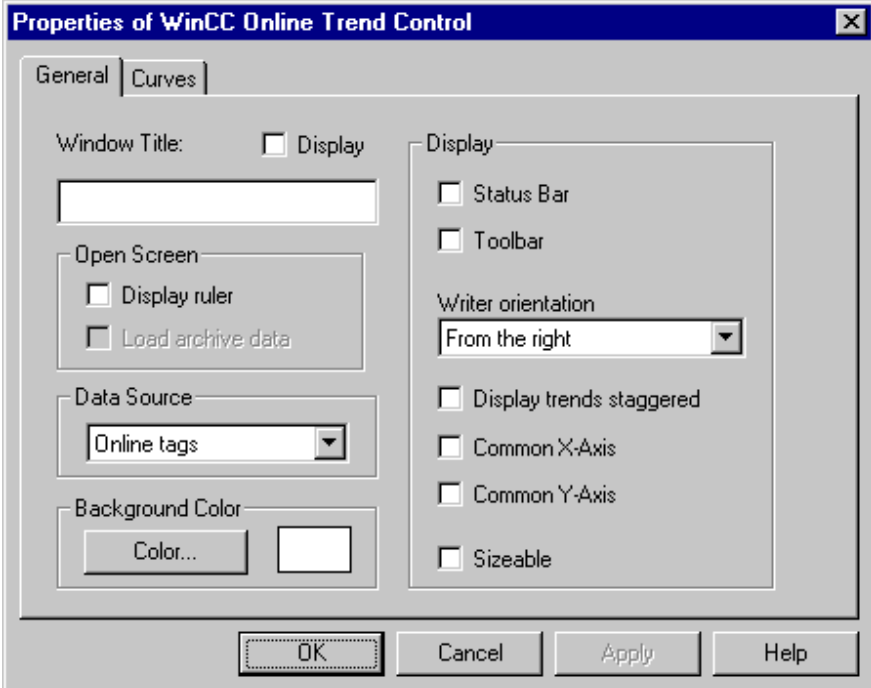
Step	Procedure: Configuration of the Table Display
1	Create a new picture in the <i>Graphics Designer</i> . In this sample, this is the <i>ex_3_chapter_01b.pdl</i> picture.
2	<p>Configuration of the Control used for displaying the table. This is the <i>WinCC Online Table Control</i>. It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture.</p>  <p>The screenshot shows the 'Objects' window with the 'Object Palette' tab active. The 'Selection' menu is open, and the 'WinCC Online Table Control' is highlighted. The palette also lists other controls: WinCC Digital/Analog Clock Control, WinCC Gauge Control, WinCC Online Trend Control, WinCC Push Button Control, WinCC Slider Control, WinCC User Archive - Table Element, WinCC Alarm Control, and WinCC DXF Control. At the bottom, there are tabs for 'Standard' and 'Controls'.</p>

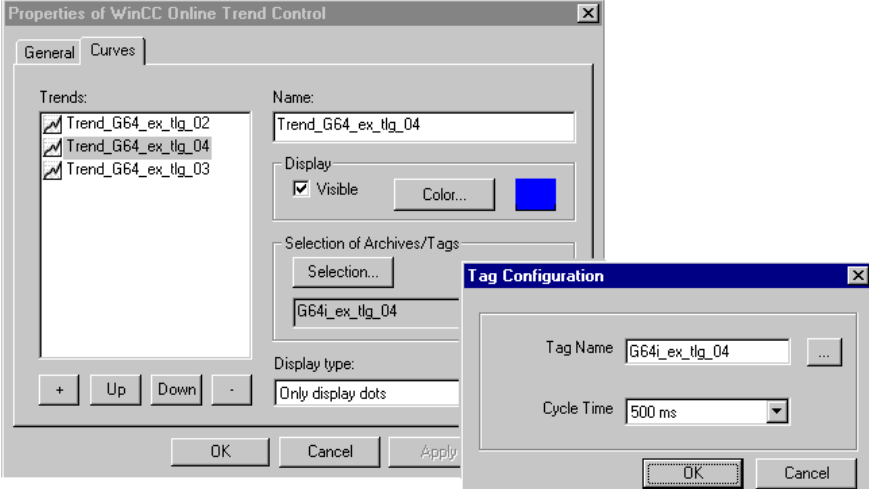
Step	Procedure: Configuration of the Table Display
3	<p>After placing the Control in the picture, the <i>WinCC Online Table Control Properties</i> dialog is automatically opened.</p> <p>In the <i>General Information</i> tab, you can specify the Control's title and how it is labeled. In this sample, the <i>Display</i> check-box is deselected. A <i>Window Title</i> is still entered. In the <i>C-Actions</i> created later, this window title is used to reference the corresponding Control. The name of the previously created archive <i>AZ_ProcessValueArchive_00</i> is used.</p> <p>Via the <i>Color</i> button, the <i>Background Color</i> of the table window is set to white.</p> <p>In the <i>Display</i> field, all check-boxes are deselected in this sample. As a result, no <i>Toolbar</i> and no <i>Status Bar</i> is displayed.</p> 


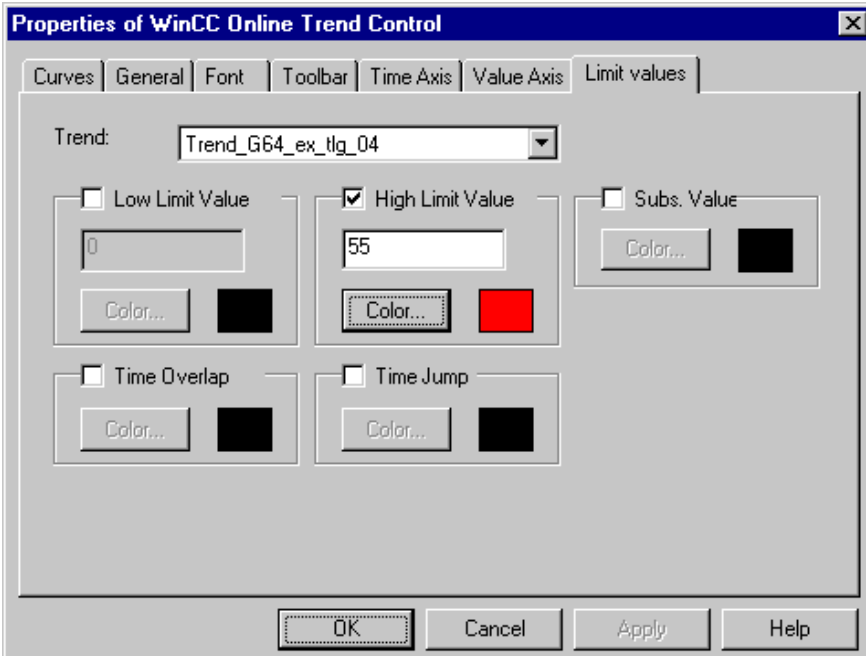
Step	Procedure: Configuration of the Table Display
4	<p>In the <i>Columns</i> tab, the columns to be displayed are specified in detail. For this sample, only one column is needed.</p> <p>One column has already been created. This column is renamed to <i>Row_G64i_ex_tlg_08</i>.</p> <p>Via the <i>Selection</i> button, the <i>Archive Tag</i> to be displayed can be assigned to the column. In this sample, the <i>Archive Tag G64i_ex_tlg_08</i> of the previously created <i>AZ_ProcessValueArchive_00</i> archive is assigned to the column.</p> <p>The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 

Step	Procedure: Configuration of the Table Display
5	<p>Specific properties settings of the column. For this purpose, an expanded properties dialog is available. This dialog is opened via a  on the Control. The properties dialog described previously, on the other hand, is opened via a  on the Control while the CTRL key is pressed.</p> <p>The expanded properties dialog contains in addition to the already mentioned <i>General Information</i> and <i>Columns</i> tabs three additional tabs. In this sample, settings are made only in the <i>Column</i> tab.</p> <p>The <i>Format</i> of the <i>Time Display</i> is changed to <i>hh:mm:ss</i>. The <i>Orientation</i> of the <i>Time Display</i> and the <i>Value Display</i> is set to <i>Centered</i>. In the <i>Time Selection</i> field, the <i>Time Range</i> check-box remains selected, but the set <i>Range</i> is changed to <i>1 X 1 Hour</i>.</p> <p>For the remaining options, the presettings are kept. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 
6	<p>Writing to a tag if the set value is exceeded. This is done via a C-Action at the Control at  → <i>Properties</i> → <i>Geometry</i> → <i>Position X</i>. The C-Action has been created at the property, since a trigger is required. The property itself is not made dynamic.</p>



Configuration of the Trend Display



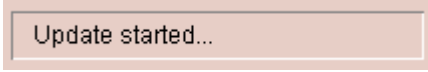
Step	Procedure: Configuration of the Trend Display
1	<p>Configuration of the Control used for displaying the trend profile. This is the <i>WinCC Online Trend Control</i>. It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture. After placing the Control in the picture, its configuration dialog will be opened automatically.</p> <p>In the <i>General Information</i> tab, specify that the Control is to be displayed without a title bar. For this purpose, the <i>Display</i> check-box is deselected.</p> <p>In the <i>Data Supply</i> field, <i>Online Tag</i> is set. In this way, no separate archive must be created to display the chronological progress of a tag. Just set the desired internal or external tag. The temporary buffering of the tag values needed for the display is performed by the Control itself.</p> <p>Via the <i>Color</i> button, the <i>Background Color</i> of the trend window is set to white.</p> <p>In the <i>Display</i> field, all check-boxes are deselected.</p> 

Step	Procedure: Configuration of the Trend Display
2	<p>In the <i>Trends</i> tab, the trend profile to be displayed is specified in detail.</p> <p>The trend is renamed to <i>Trend_G64i_ex_tlg_04</i>. The <i>Color</i> of the trend is set to <i>blue</i>. As the <i>Display Type</i>, <i>Fill Area Interpolate Trend</i> is set.</p> <p>Via the <i>Selection</i> button, the <i>Tag Configuration</i> dialog is opened. In this dialog, the tag to be displayed is set. In this case, this not an <i>Archive Tag</i>, but a so-called <i>Online Tag</i> (an internal or external tag). In addition, a <i>Cycle Time</i> is set. In this sample, the <i>G64i_ex_tlg_04</i> tag with a cycle of <i>500 ms</i> is set.</p> <p>The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p>  <p>The screenshot shows two overlapping dialog boxes. The background dialog is titled 'Properties of WinCC Online Trend Control' and has two tabs: 'General' and 'Curves'. The 'General' tab is active. It features a 'Trends' list box containing three entries: 'Trend_G64_ex_tlg_02', 'Trend_G64_ex_tlg_04', and 'Trend_G64_ex_tlg_03'. To the right of this list is a 'Name' text box containing 'Trend_G64_ex_tlg_04'. Below the name box is a 'Display' section with a checked 'Visible' checkbox and a 'Color...' button next to a blue color swatch. Further down is a 'Selection of Archives/Tags' section with a 'Selection...' button and a text box containing 'G64i_ex_tlg_04'. At the bottom of this dialog is a 'Display type' dropdown menu set to 'Only display dots'. The foreground dialog is titled 'Tag Configuration' and has a 'Tag Name' text box containing 'G64i_ex_tlg_04' and a 'Cycle Time' dropdown menu set to '500 ms'. Both dialogs have 'OK' and 'Cancel' buttons at the bottom.</p>

Step	Procedure: Configuration of the Trend Display
3	<p>Colored identification of a limit value violation. This can only be done in the expanded properties dialog. This dialog is opened via a  on the Control. The expanded properties dialog contains in addition to the already mentioned <i>General Information</i> and <i>Trends</i> tabs five additional tabs.</p> <p>In the <i>Value Axis</i> tab, the <i>Automatic</i> check-box is deselected in the <i>Range Selection</i> field. The range is set fixed from <i>-100</i> to <i>100</i>.</p> <p>In the <i>Limit Values</i> tab, a <i>High Limit Value</i> is configured. This value is set to <i>55</i>. The color of the limit value is set to red.</p> <p>The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 

Configuration of the Toolbar and Status Bar

Step	Procedure: Configuration of the Toolbar and Status Bar
1	<p>In Tag Management, create two internal tags of the <i>Binary Tag</i> type. In this sample, the <i>BINi_ex_tlg_06</i> and <i>BINi_ex_tlg_07</i> tags are used.</p>
2	<p>To control the update, a <i>Smart Object</i> → <i>Status Display</i> is configured. In this sample, the <i>Status Display3</i> object is used.</p> <p>Via its <i>configuration dialog</i>, the object is connected to the <i>BINi_ex_tlg_06</i> tag and triggered upon change. The stati <i>0</i> and <i>1</i> are created. In this sample, the bitmaps <i>stop tlg.bmp</i> and <i>stop go tlg.bmp</i> are assigned to these stati. The object's <i>configuration dialog</i> can be exited by clicking on <i>OK</i>.</p> <p>At the just configured <i>Status Display3</i> object, create a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>. This <i>C-Action</i> simulates the pressing of the Stop/Go button from the Control's standard toolbar. In addition, the status of the <i>BINi_ex_tlg_06</i> tag is inverted to display the changed status of the Control's update. A tag value of zero corresponds to an activated update.</p> <p>The status of the <i>BINi_ex_tlg_06</i> tag is always zero at the opening of the picture, since the update of the trend window is always activated when the picture is opened. This is implemented via a <i>C-Action</i> at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> of the Picture Object <i>ex_3_chapter_01b.pdl</i>. This sets the status of the tag to <i>0</i>.</p> 
3	<p>As described in step 2, configure a second <i>Smart Object</i> → <i>Status Display</i>. In this sample, the <i>Status Display2</i> object is used. This status display controls the editability of the table.</p> <p>The object is linked with the <i>BINi_ex_tlg_07</i> tag. Correspondingly, different bitmaps are used (<i>Edit.gif</i> / <i>Edit inv.gif</i>).</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, a <i>C-Action</i> is created. This <i>C-Action</i> simulates the pressing of the edit button of the Control's standard toolbar. In addition, the status of the <i>BINi_ex_tlg_07</i> tag is inverted to display the changed status of the table's editability. A tag value of zero corresponds to deactivated editability.</p> <p>The status of the <i>BINi_ex_tlg_07</i> tag is always zero at the opening of the picture, since the editability of the table window is always deactivated when the picture is opened. This is implemented by expanding the <i>C-Action</i> at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> of the Picture Object <i>ex_3_chapter_01b.pdl</i>. A statement is inserted, which set the status of the tag to <i>0</i>.</p> 

Step	Procedure: Configuration of the Toolbar and Status Bar
4	<p>In order to navigate in the archive while the update is stopped, replicas of the four navigation buttons of the Control's standard toolbar are needed.</p> <p>For the implementation, four <i>Windows Objects</i> → <i>Buttons</i> are configured; in this sample, these are the <i>Button4</i>, <i>Button7</i>, <i>Button8</i> and <i>Button11</i> objects.</p> <p>For each of these objects, a <i>C-Action</i> is created at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>. These actions simulate the pressing of the buttons on the standard toolbar.</p> <p>in addition, a <i>Smart Object</i> → <i>Graphic Object</i> is required that places itself over these buttons and makes them inoperational in case the update is started. In this sample, this is the <i>Graphic Object2</i>. This object displays the four buttons in an inoperational status (<i>Pfeile dis.bmp</i>). At <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i>, create a <i>Dynamic Dialog</i>. This dialog controls the visibility of the object dependent on the <i>BINi_ex_tlg_06</i> tag, which contains information about the update of the Control.</p> 
5	<p>An additional <i>Smart Object</i> → <i>Graphic Object</i> is required. This object is used to make the Stop/Go button inoperational if the editability of the table is activated. In this sample, the <i>Graphic Object1</i> is used.</p> <p>At <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i>, create a <i>Dynamic Dialog</i> which makes the object visible if the <i>BINi_ex_tlg_07</i> tag receives the <i>TRUE</i> status, i.e. the table can be edited. As the picture to be displayed, this sample uses <i>stop dis tlg.bmp</i>. This object must be positioned exactly on top of the Stop/Go button.</p> 
6	<p>For the display of the status bar, a <i>Windows Object</i> → <i>Button</i> is configured. In this sample, this is the <i>Button10</i> object.</p> <p>For the text display, a <i>Button</i> is used since it can easily be equipped with a 3D border. For this purpose, therefore, no additional objects are required.</p> <p>For <i>Button10</i>, create a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Font</i> → <i>Text</i>. This dialog either returns the text <i>Update Started</i> or <i>Update Stopped</i> to the property dependent on the <i>BINi_ex_tlg_06</i> tag.</p> 

C-Action at the WinCC Online Table Control (Control1)

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    if (GetTagDouble("G64i_ex_tlg_04") >= 55)
    {
        SetTagDouble("G64i_exTlg_08", GetTagDouble("G64i_ex_tlg_04"));
    }

    return 50;
}
```

- Reading of the G64I_ex_tlg_04 tag and checking whether the current value is greater than 55.
- If the value is greater than 55, writing of the value to the D64I_ex_tlg_08 tag.

C-Action at the Edit Button (Status Display2)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz)
{
    TlgTableWindowPressEditRecordButton("AZ_ProcessValueArchive_00");
    SetTagBit("BINi_ex_tlg_07", (SHORT)!GetTagBit("BINi_ex_tlg_07"));
    SetTagBit("BINi_ex_tlg_06", TRUE);
}
```

- The call of the standard function *TlgTableWindowPressEditButton* has the same effect as pressing the edit button on the Control's standard toolbar. A text is assigned to the function to allow it to identify the Control to be accessed. This text is the window title that has been specified during the configuration of the Control. In this sample, this was the text *AZ_ProcessValueArchive_00*.
- Inverting the *BINi_ex_tlg_07* tag to store the current status of the table's editability.
- Setting the *BINi_ex_tlg_06* tag to true to stop the update.

C-Action at the Navigation Button Start (Button4)

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
    TlgTableWindowPressFirstButton("AZ_ProcessValueArchive_00");
}
```

- The call of this standard function has the same effect as pressing the *First Data Record* button on the standard toolbar. The functions used at the other buttons are:
 - *TlgTrendWindowPressPrevButton*
 - *TlgTrendWindowPressNextButton*
 - *TlgTrendWindowPressLastButton*
 - *TlgTableWindowPressOpenTimeSelectDlgButton*
 - *TlgTableWindowPressStartStopButton*

Note:

For each button on the standard toolbar of the *WinCC Online Table Control*, a corresponding standard function is available which simulates the pressing of each button.

Note for the General Application

The following adaptations must be made before the general application:

- The tags to be archived must be adapted to meet your own requirements.
- The event that initiates the archiving must be defined by the user. For this purpose, a project function must be created.
- The appearance of the required elements can be adapted to meet your own needs. The same applies to the status bar.

4.1.4 User-Defined Table Layout (ex_3_chapter_01c.pdl)

Task Definition

A process value is to be acquired cyclically. For a time periods lasting 10 seconds, the mean value, maximum value and minimum value are to be determined. These values are to be stored in the project-internal archive.

The stored values are to be displayed in a table. This table is user-defined in the *Graphics Designer* editor. This is necessary, if a table layout is needed that cannot be implemented using the standard tools of *Tag Logging*.

Implementation Concept

To archive the data, a cyclic-continuous process value archive is created in the *Tag Logging* editor.

For the implementation of the graphical display, either a *Standard Object* → *Static Text* or a *Smart Object* → *I/O Field* is used for each table line, depending on the type of information to be displayed.

The data is read from the database table of the corresponding archive via API functions.

Creating a Process Value Archive

Step	Procedure: Creating a Process Value Archive
1	In Tag Management, create three tags of the <i>Floating-Point Number 64-Bit IEEE 754</i> type. In this sample, these are the <i>G64i_ex_tlg_05</i> , <i>G64i_ex_tlg_06</i> and <i>G64i_ex_tlg_07</i> tags. Into these tags, the archived values are written as well.
2	Create a new <i>Process Value Archive</i> using the Archive Wizard. In this sample, the archive has been named <i>ZK_ProcessValueArchive_02</i> . As the tag to be archived, the <i>G64i_ex_tlg_04</i> tag has been selected three times.
3	In the properties dialog of the process value archive, the size of the archive is set to <i>100</i> data records. For the remaining options, the default settings are kept.
4	In the properties dialog of the first process tag, <i>MaximumValue</i> is entered as the <i>Name of the Archive Tag</i> in the <i>General Information</i> tab. In the <i>Also write Archived Value to Tag</i> field, the tag <i>G64i_ex_tlg_07</i> is set. Via this tag, a <i>C-Action</i> can be used to react to the archiving of a value. This can be accomplished, by triggering this <i>C-Action</i> upon the change of the tag. In the <i>Parameters</i> tab, set the <i>Acquisition</i> in the <i>Cycle</i> field to <i>500 ms</i> and the <i>Archiving</i> to <i>20*500 ms</i> . In the <i>Processing</i> field, the <i>Maximum Value</i> radio-button is selected. This means that the selected tag is acquired every 500 ms and archived every 10 s. The value archived is the largest value that occurred during the 10 s period. For the remaining options in the dialog, the default settings are kept.


Processing


Actual value
 Sum
 Maximum value
 Mean value
 Action
 Minimum value

Step	Procedure: Creating a Process Value Archive
5	For the two other archive tags, the settings are made following steps 3 and 4. As the <i>Processing</i> , however, the <i>Minimum Value</i> and <i>Mean Value</i> radio-buttons are selected. The <i>Name of the Archive Tag</i> is assigned accordingly.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	In Tag Management, create two internal tags of the <i>Binary Tag</i> type. In this sample, these are the <i>BINi_ex_tlg_06</i> and <i>FLAG_TableGetOutputValue</i> tags.
2	<p>Creation of a project function that instructs <i>Tag Logging</i> to transfer the archive data to another function (Callback function). This function is called once for every data record and contains information about that data record in the form of a special structure type. The transferred data is stored in a static array of this structure type.</p> <p>In this sample, the functions <i>EnumerateSuperArchiveData</i> and <i>GetArchiveDataCallback</i> are used.</p> <p>The sample uses two external C variables.</p> <ul style="list-style-type: none"> • extern int dwSize • extern WORD wOffset <p>They must be created at the start of the project. For this purpose, a separate project function is used. Its call is inserted into the <i>C-Action at Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> of the start picture <i>ex_0_startpicture_00.PDL</i>. In the sample, this function is called <i>CreateExternal</i>.</p>

Step	Procedure: Implementation in the Graphics Designer																																												
3	<p>Create a new picture, in the sample this is the <i>ex_3_chapter_01b.pdl</i> picture. The user-defined is to display 10 lines. For the display of the data in the first column, 10 <i>Standard Objects</i> → <i>Static Texts</i> are used, which display the date and time. For the additional columns, <i>Smart Objects</i> → <i>I/O Fields</i> are used. As the object names in the first column, the names <i>Static Text1</i> to <i>Static Text10</i> are used, where the number defines the line. The numbering is performed from bottom to top, since the last line contains the latest data.</p> <p>The <i>I/O Fields</i> use a number code as name. The first digit indicates the column number, the second number the line number.</p> <table border="1" data-bbox="483 615 1343 1213"> <thead> <tr> <th>Date/Time</th> <th>Mean Value</th> <th>Minimum</th> <th>Maximum</th> </tr> </thead> <tbody> <tr><td>06.07.1999 10:17:27</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:17:37</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:17:47</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:17:57</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:18:07</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:18:17</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:18:27</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:18:37</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:18:47</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> <tr><td>06.07.1999 10:18:57</td><td>000,00</td><td>000,00</td><td>000,00</td></tr> </tbody> </table>	Date/Time	Mean Value	Minimum	Maximum	06.07.1999 10:17:27	000,00	000,00	000,00	06.07.1999 10:17:37	000,00	000,00	000,00	06.07.1999 10:17:47	000,00	000,00	000,00	06.07.1999 10:17:57	000,00	000,00	000,00	06.07.1999 10:18:07	000,00	000,00	000,00	06.07.1999 10:18:17	000,00	000,00	000,00	06.07.1999 10:18:27	000,00	000,00	000,00	06.07.1999 10:18:37	000,00	000,00	000,00	06.07.1999 10:18:47	000,00	000,00	000,00	06.07.1999 10:18:57	000,00	000,00	000,00
Date/Time	Mean Value	Minimum	Maximum																																										
06.07.1999 10:17:27	000,00	000,00	000,00																																										
06.07.1999 10:17:37	000,00	000,00	000,00																																										
06.07.1999 10:17:47	000,00	000,00	000,00																																										
06.07.1999 10:17:57	000,00	000,00	000,00																																										
06.07.1999 10:18:07	000,00	000,00	000,00																																										
06.07.1999 10:18:17	000,00	000,00	000,00																																										
06.07.1999 10:18:27	000,00	000,00	000,00																																										
06.07.1999 10:18:37	000,00	000,00	000,00																																										
06.07.1999 10:18:47	000,00	000,00	000,00																																										
06.07.1999 10:18:57	000,00	000,00	000,00																																										
4	<p>For each <i>Static Text</i>, a <i>C-Action</i> is created at <i>Properties</i> → <i>Font</i> → <i>Text</i>. This action reads the date to be displayed from the Callback function with respect to its own object number. The function is triggered upon the change of the <i>FLAG_TableGetOutputValue</i> tag. The status of this tag changes, if the archive has received new data and this data has been read.</p> <p>In the same manner, create a <i>C-Action</i> for each <i>I/O Field</i> at <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i>. This action has also the task to read the data record assigned to an object from the Callback function.</p>																																												
5	<p>To control the update, a <i>Smart Object</i> → <i>Status Display</i> is configured. In this sample, the <i>Status Display3</i> object is used.</p> <p>Via its <i>configuration dialog</i>, the object is connected to the <i>BINi_ex_tlg_06</i> tag and triggered upon change. The stati <i>0</i> and <i>1</i> are created and corresponding pictures assigned to each status. In this sample, the bitmaps <i>stop go.tlg.gif</i> and <i>stop.tlg.gif</i> are used.</p> 																																												

Step	Procedure: Implementation in the Graphics Designer
6	<p>At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, create a <i>C-Action</i> that inverts the status of the <i>BINi_ex_tlg_06</i> tag. The status <i>TRUE</i> means that the update has been started.</p> <p>The status of this tag is <i>TRUE</i> at the opening of the picture, since the update of the table window is always started when the picture is opened. This is implemented via a <i>C-Action</i> at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> of the Picture Object <i>ex_3_chapter_01c.pdl</i>. This <i>C-Action</i> sets the status of the tag to <i>TRUE</i> and reads the archive one time.</p>
7	<p>For the <i>Statusdisplay3</i>, create a <i>C-Action</i> at <i>Properties</i> → <i>Geometry</i> → <i>Width</i>. This action, dependent on the status of the <i>BINi_ex_tlg_06</i> tag, reads the archive and inverts the <i>FLAG_TableGetOutputValue</i> tag to trigger the update of the table. This <i>C-Action</i> is triggered upon the change of the <i>G64i_ex_tlg_07</i> tag, in which the archived value is stored as well. The action also reacts to the next archiving if the value is identical to the previous one. For this purpose, the <i>G64i_ex_tlg_07</i> tag is set to a value that cannot be reached by the process value to be archived after every run of the <i>C-Action</i>.</p>
8	<p>To navigate in the archive while the update is stopped, navigation buttons are needed.</p> <p>For the implementation, four <i>Windows Objects</i> → <i>Buttons</i> are configured; in this sample, these are the <i>Button4</i>, <i>Button7</i>, <i>Button8</i> and <i>Button11</i> objects.</p> 
9	<p>At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, <i>C-Actions</i> are created. These actions write a new value to the external C variable <i>dwOffset</i>. In addition the trigger tag <i>FLAG_TableGetOutputValue</i> is inverted to achieve an update of the display.</p> <p>in addition, a <i>Smart Object</i> → <i>Graphic Object</i> is required that places itself over these buttons and makes them inoperational in case the update is started. In this sample, this is performed by the <i>Graphic Object2</i>. The bitmap displayed by this object shows the four buttons in an inoperational status (<i>Pfeile dis.bmp</i>). At <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i>, create a <i>tag connection</i> to the <i>BINi_ex_tlg_06</i> tag that is triggered upon change.</p>

Project Function for Reading the Archive

```

#include "apdefap.h"
BOOL EnumerateSuperArchiveData()
{
extern DWORD    dwSize;
BOOL           fRet;
TLG_GETARCHIVDATA    GAD;
CMN_ERROR           Error;
LPTSTR lpszArchivName = "ZK_ProcessValueArchive_02";
LPTSTR lpszVarName3 = "MaximumValue" ;
LPTSTR lpszVarName2 = "MinimumValue" ;
LPTSTR lpszVarName1 = "MeanValue" ;
SYSTEMTIME    sysFrom;
SYSTEMTIME    sysTo;
time_t Time;
struct tm*    TimeStruct;

time(&Time);
TimeStruct = localtime(&Time);

sysTo.wYear    = (WORD)(TimeStruct->tm_year+1900);
sysTo.wMonth   = (WORD)(TimeStruct->tm_mon+1);
sysTo.wDay     = (WORD)(TimeStruct->tm_mday);
sysTo.wHour    = (WORD)(TimeStruct->tm_hour);
sysTo.wMinute  = (WORD)(TimeStruct->tm_min);
sysTo.wSecond  = (WORD)(TimeStruct->tm_sec);

sysFrom.wYear  = 1997;
sysFrom.wMonth = 1;
sysFrom.wDay   = 1;
sysFrom.wHour  = 0;
sysFrom.wMinute = 0;
sysFrom.wSecond = 0;

Call(&GAD, (PVOID)0);

if (TLGConnect( NULL , &Error )==FALSE) {
    printf("Error: %s\r\n",Error.szErrorText);
    return FALSE;
}
else {
    fRet=TLGGetArchivData(lpszArchivName,lpszVarName1,
        sysFrom,sysTo,GetArchiveDataCallback,
        (PVOID)1,0,&Error);
    if (fRet==FALSE)
        printf("Error: %s\r\n",Error.szErrorText);

    fRet=TLGGetArchivData(lpszArchivName,lpszVarName2,
        sysFrom,sysTo,GetArchiveDataCallback,
        (PVOID)2,0,&Error);
    if (fRet==FALSE)
        printf("Error: %s\r\n",Error.szErrorText);

    fRet=TLGGetArchivData(lpszArchivName,lpszVarName3,
        sysFrom,sysTo,GetArchiveDataCallback,
        (PVOID)3,0,&Error);
    if (fRet==FALSE)
        printf("Error: %s\r\n",Error.szErrorText);

    Call(&GAD, (PVOID)4);
    dwSize=GAD.dwFlags;
    TLGDisconnect( NULL );
    return TRUE;
}
}

```


- Definition of the values for the start and end time, between which the data is read from the archive. As the start value, a fixed time is set, as the end time the current system time.
- Initialization of the Callback function via the help function *Call*. This function calls the *GetArchiveDataCallback* function with a value of 0 for the *lpUser* parameter.
- Establishing the connection to *Tag Logging*. If this fails, the function is aborted.
- Reading the values archived from the *MaxValue*, *MinValue* and *MeanValue* archive tags via the function *TLGGetArchiveData*.
- Determination of the number of values read. This is done via the help function *Call*, which calls the function *GetArchiveDataCallback* with a value of 4 for the *lpUser* parameter.
- Termination of the connection to *Tag Logging*.

Callback Function

```

BOOL GetArchiveDataCallback (PTLG_GETARCHIVDATA    lpGAD, PVOID    lpUser)
{
    static int i1 = 0;
    static int i2 = 0;
    static int i3 = 0;

    WORD wRecordNumber;
    WORD wColumnNumber;

    static TLG_GETARCHIVDATA GAD1[100];
    static TLG_GETARCHIVDATA GAD2[100];
    static TLG_GETARCHIVDATA GAD3[100];

    int User;
    User=(int)lpUser;

    if ((User==1)||((User==2)||((User==3)))
    {
        switch(User)
        {
        case 1 : if (i1<=100) {GAD1[i1]=*lpGAD; i1++;} break;
        case 2 : if (i2<=100) {GAD2[i2]=*lpGAD; i2++;} break;
        case 3 : if (i3<=100) {GAD3[i3]=*lpGAD; i3++;} break;
        }//switch

    }//if ((User==1)||((User==2)||((User==3)))

    if (User==0)
    {
        i1=0;
        i2=0;
        i3=0;

        memset(&GAD1[0],0,sizeof(TLG_GETARCHIVDATA)*100);
        memset(&GAD2[0],0,sizeof(TLG_GETARCHIVDATA)*100);
        memset(&GAD3[0],0,sizeof(TLG_GETARCHIVDATA)*100);

    }//if (User==0)

    if (User==4)
    {
        lpGAD->dwFlags=i1;
    }//if (User==4)

    if (User==7)
    {
        wRecordNumber=lpGAD->stTime.wMonth;
        wColumnNumber=lpGAD->stTime.wDay;

        switch(wColumnNumber)
        {
        case 0 : lpGAD->stTime.wYear = GAD1[wRecordNumber].stTime.wYear;
                lpGAD->stTime.wMonth = GAD1[wRecordNumber].stTime.wMonth;
                lpGAD->stTime.wDay = GAD1[wRecordNumber].stTime.wDay;
                lpGAD->stTime.wHour = GAD1[wRecordNumber].stTime.wHour;
                lpGAD->stTime.wMinute = GAD1[wRecordNumber].stTime.wMinute;
                lpGAD->stTime.wSecond = GAD1[wRecordNumber].stTime.wSecond;
                break;
        case 1 : lpGAD->doValue=GAD1[wRecordNumber].doValue;
                break;
        case 2 : lpGAD->doValue=GAD2[wRecordNumber].doValue;
                break;
        case 3 : lpGAD->doValue=GAD3[wRecordNumber].doValue;
                break;
        default : break;
        }//switch

    }//if (User==7)

    return TRUE;
}

```

- Declaration of three static counter tags: i1, i2 and i3.
- Declaration of three static arrays consisting of structures of the TLG_GETARCHIVEDATA type. In these arrays, the archive values are stored.
- If the transfer parameter *lpUser* has a value of 1, 2 or 3, then the function has been called by Tag Logging. In this case, the transferred structure *lpGAD* is stored in the corresponding array.
- If the transfer parameter *lpUser* has the value of 0, then it is an initialization run. The counter tags are reset to 0 and memory space is reserved for the arrays.
- If the transfer parameter *lpUser* has the value of 4, then the number of stored values is requested. This is stored in the transferred structure as the structure member *dwFlags*.
- If the transfer parameter *lpUser* has the value of 7, then the value of a stored tag in an *I/O Field* or *Static Text* is requested. From which position of the table the tag has been requested, is specified in the transferred structure by the structure members *stTime.wMonth* and *stTime.wDay*.

C-Action at the Static Texts

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    int nRecordNumber;
    WORD wColumnNumber;
    extern WORD wOffset;
    char szObject[5];
    TLG_GETARCHIVEDATA GAD;
    PVOID lpUser7 = 7;
    char szTime[20] = "";
    int nObjectNumber;
    extern DWORD dwSize;

    wColumnNumber=0;

    nObjectNumber=atoi(lpszObjectName+15);
    nRecordNumber=(dwSize-nObjectNumber-wOffset);

    if (nRecordNumber<0) return "";

    GAD.stTime.wMonth=(WORD)nRecordNumber;
    GAD.stTime.wDay=wColumnNumber;

    GetArchiveDataCallback(&GAD, lpUser7);

    sprintf(szTime, "%02d.%02d.%d %02d:%02d:%02d",
        GAD.stTime.wDay, GAD.stTime.wMonth, GAD.stTime.wYear, GAD.stTime.wHour,
        GAD.stTime.wMinute, GAD.stTime.wSecond);

    return szTime;
}
```

- The structure members *stTime.wMonth* and *stTime.wDay* of the *GAD* structure to be transferred are supplied with the column number or the calculated data record number. The object name contains information about the data record number.
- The *GetArchiveDataCallback* function is called by the value 7 of the transfer parameter *lpUser*, i.e. a value is requested.
- The date value is stored in the structure member *stTime* of the transferred structure *GAD*. From this, the text to be displayed is formed. This text is returned to the property via *return*.

C-Action at the I/O Fields

```
#include "apdefap.h"
double _main(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

extern dwSize;
int nRecordNumber;
WORD wColumnNumber;
extern WORD wOffset;
char szObject[5];
TIG_GETARCHIVDATA GAD;
PVOID lpUser = 7;

strcpy(szObject, "");
sprintf(szObject, "%c", lpszObjectName[0]);
wColumnNumber=(WORD)atoi(szObject);

nRecordNumber=(dwSize-atoi(lpszObjectName+1)-wOffset);

if (nRecordNumber<0) return 0;

GAD.stTime.wMonth=(WORD)nRecordNumber;
GAD.stTime.wDay=wColumnNumber;

GetArchiveDataCallback(&GAD, lpUser);

return GAD.doValue;

}
```

- From the object name, the column and line number is determined. The structure members *stTime.wMonth* and *stTime.wDay* of the *GAD* structure to be transferred are supplied with the column number or the calculated data record number.
- The *GetArchiveDataCallback* function is called by the value 7 of the transfer parameter *lpUser*, i.e. a value is requested.
- The tag value is stored in the structure member *doValue* of the transferred structure *GAD* and is used as the return value.

Note:

The toolbar contains a button displayed below for setting the parameters of the table. Via this button, the dialog for setting the colors of the various table elements is accessed. A brief description about this can be found in the sample Color Dialogs (ex_3_chapter_01c).




Note for the General Application

The following adaptations must be made before the general application:

- The archive data to be displayed must be adapted to meet your own requirements.
- Adapt the table layout to meet your own requirements. If a different column or line number is needed, the C-Actions and project functions must be adapted.

4.1.5 Archiving Binary Tags (ex_3_chapter_01d.pdl)

Task Definition

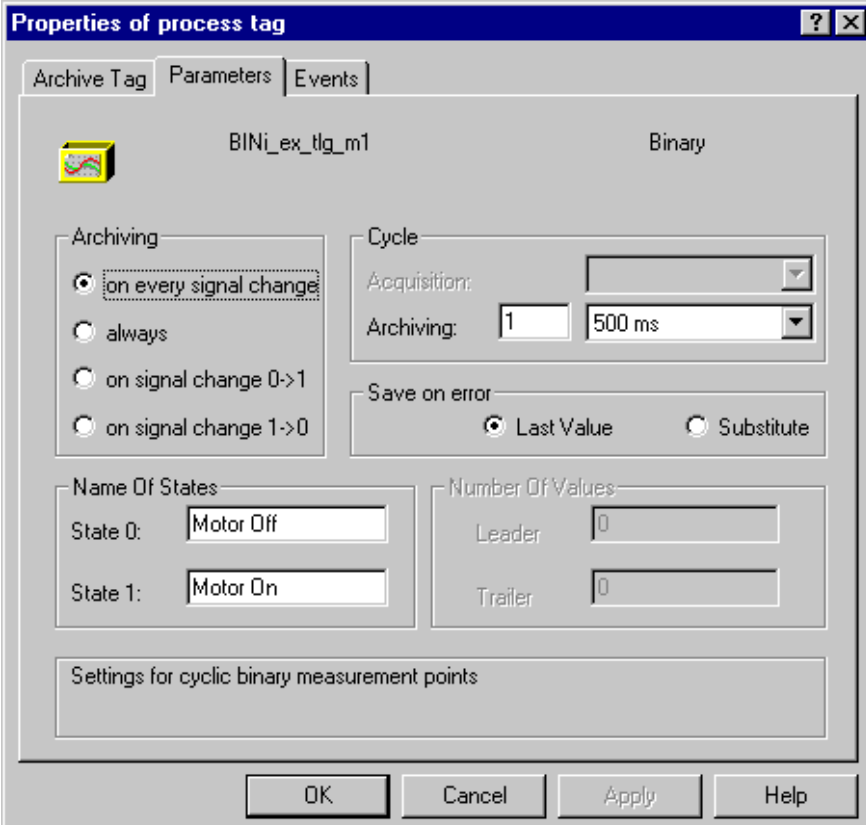
The switching operations of the three motors are to be stored in an archive. If a motor is selected using the , a table is displayed showing the last switching operations in a one day period. The table only displays the status of the selected motor.

Implementation Concept

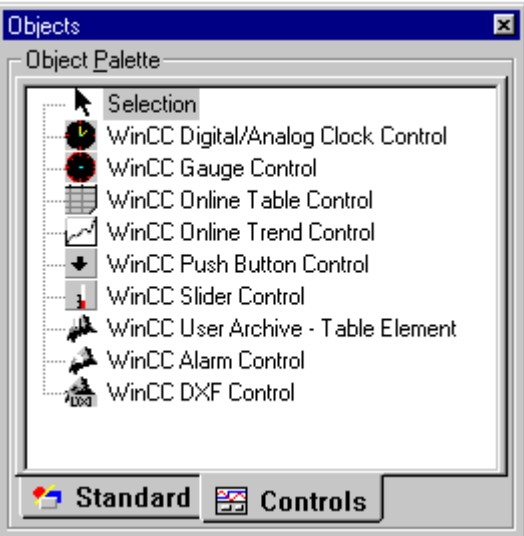
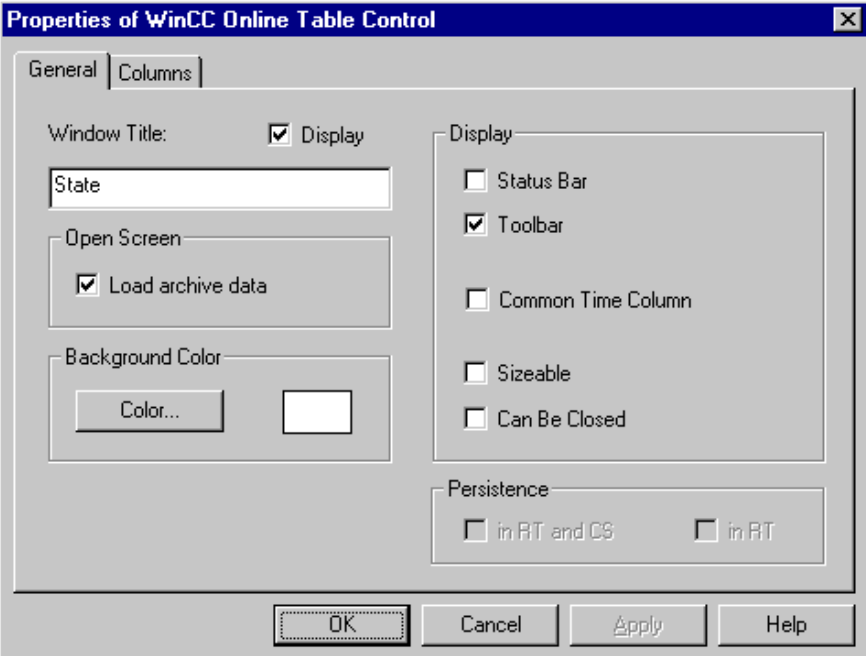
To archive the data to be displayed, a cyclic-continuous process value archive is created in the *Tag Logging* editor. Each tag is displayed in a separate column. For the implementation of the graphical display, a WinCC Online Table Control is used that displays the appropriate column according to the motor selected.

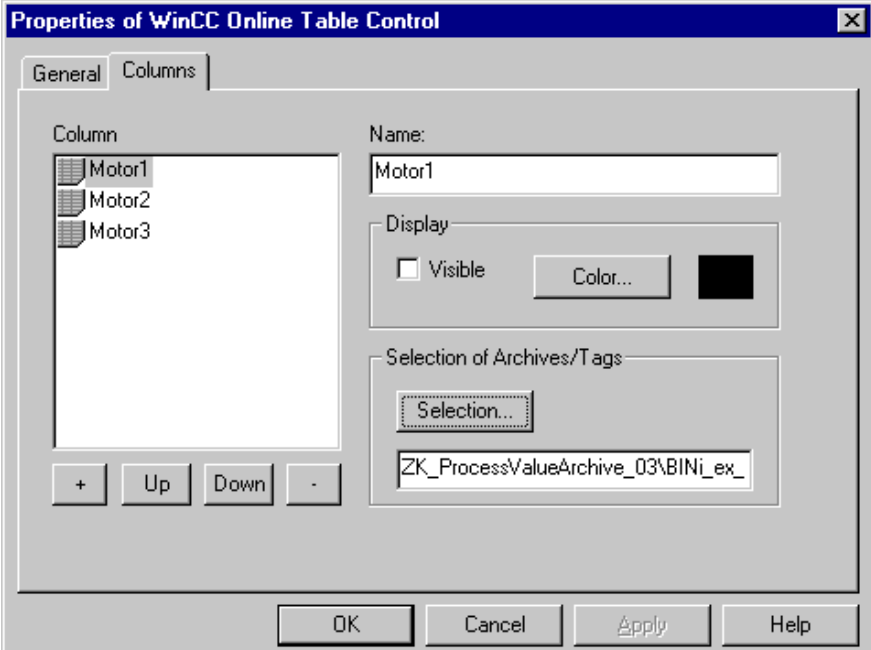
Creating a Process Value Archive



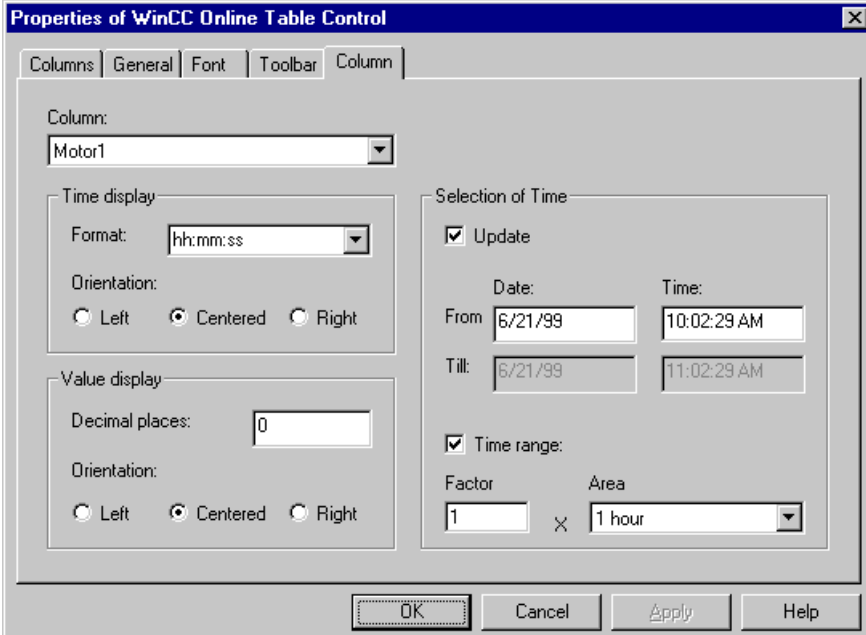
Step	Procedure: Creating a Process Value Archive
1	In Tag Management, create four tags. In this sample, these are the <i>BINi_ex_tlg_m1</i> , <i>BINi_ex_tlg_m2</i> and <i>BINi_ex_tlg_m3</i> tags of the <i>Binary Tag</i> type and the <i>U08i_ex_tlg_00</i> tag of the <i>Unsigned 8-Bit Value</i> type.
2	Create a new <i>Process Value Archive</i> using the Archive Wizard. In this sample, the archive has been named <i>ZK_ProcessValueArchive_03</i> . For archiving, the <i>BINi_ex_tlg_m1</i> , <i>BINi_ex_tlg_m2</i> and <i>BINi_ex_tlg_m3</i> tags are selected.
3	In the properties dialog of the process value archive, the size of the archive is set to 40 data records. For the remaining options, the default settings are kept.

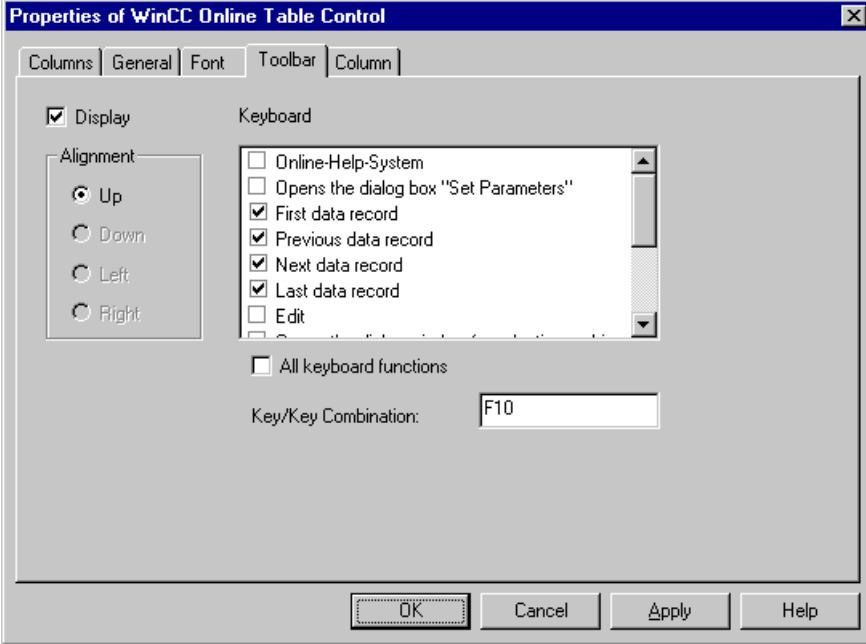
Step	Procedure: Creating a Process Value Archive
4	<p>In the properties dialog of the first process tag, in the <i>Archive upon</i> field of the <i>Parameters</i> tab, select the entry <i>Every Signal Change</i>. In the <i>Name of the Status</i> field, <i>Motor Off</i> is entered for <i>Status 0</i> and <i>Motor On</i> for <i>Status 1</i>. As the <i>Cycle</i>, <i>Archiving 1*500 ms</i> is set.</p> <p>For the remaining options, the default settings are kept.</p> <p>For the remaining archive tags, the same settings are performed.</p> 

Configuration of the Table Display

Step	Procedure: Configuration of the Table Display
1	Create a new picture, in the sample this is the <i>ex_3_chapter_01d.pdl</i> picture.
2	<p>Configuration of the Control used for displaying the table. This is the <i>WinCC Online Table Control</i>. It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture.</p> 
3	<p>After placing the Control in the picture, the <i>WinCC Online Table Control Properties</i> dialog is automatically opened.</p> <p>In the <i>General Information</i> tab, you can specify the Control's title and how it is labeled. As the window title, Status is entered. Via the <i>Color</i> button, the <i>Background Color</i> of the table window is set to white.</p> <p>In the <i>Display</i> field, the <i>Sizeable</i> and <i>Toolbar</i> check-boxes are deselected.</p> 

Step	Procedure: Configuration of the Table Display
4	<p>In the <i>Columns</i> tab, the columns to be displayed are specified in detail. For this sample, three columns are needed.</p> <p>One column has already been created. It is renamed to Motor1.</p> <p>Via the <i>Selection</i> button, the <i>Archive Tag</i> to be displayed can be assigned to the column. In this sample, the <i>Archive Tag</i> <i>BINi_ex_tlg_m1</i> of the previously created <i>ZK_ProcessValueArchive_03</i> archive is assigned to the column.</p> <p>Add two more columns. They are assigned the <i>BINi_ex_tlg_m2</i> and <i>BINi_ex_tlg_m3</i> tags, and their column names and colors are adapted. The <i>Display Visible</i> check-box is deselected for all three columns.</p> 

Step	Procedure: Configuration of the Table Display
5	<p>Specific properties settings of the column. For this purpose, an expanded properties dialog is available. This dialog is opened via a  on the Control. The properties dialog described previously, on the other hand, is opened via a  on the Control while the CTRL key is pressed.</p> <p>The expanded properties dialog contains in addition to the already mentioned <i>General Information</i> and <i>Columns</i> tabs three additional tabs. In the <i>Column</i> tab, the following settings are made.</p> <p>The <i>Format</i> of the <i>Time Display</i> is set to <i>hh:mm:ss</i>. The <i>Orientation</i> of the <i>Time Display</i> and the <i>Value Display</i> is set to <i>Centered</i> and the decimal places to 0. In the <i>Time Selection</i> field, the <i>Time Range</i> check-box remains selected, but the set <i>Range</i> is changed to <i>1 X 1 Hour</i>.</p> 

Step	Procedure: Configuration of the Table Display
6	<p>In the <i>Toolbar</i> tab, the following check-boxes are selected at the <i>Key Functions</i> sub-entry:</p> <ul style="list-style-type: none"> • First Data Record • Previous Data Record • Next Data Record • Last Data Record • Select Time Range • Start/Stop the Update 
7	<p>Since only one column is displayed, the font size can be set to 13.5 in the <i>Font</i> tab for increased legibility.</p> <p>For the remaining options, the presettings are kept. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p>

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	<p>The motors to be displayed each consist of a <i>Standard Object</i> → <i>Circle</i>, a <i>Standard Object</i> → <i>Polygon</i> and a <i>Standard Object</i> → <i>Static Text</i>. The background color of the circle is changed according to the motor status via a <i>Dynamic Dialog</i>.</p> <p>These three objects are grouped. In the sample, this results in the creation of the <i>Group1</i>, <i>Group2</i> and <i>Group3</i> objects. Each of these objects receives a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> that writes the number of the motor to the <i>U08i_ex_tlg_00</i> tag and a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Press Left</i> that makes the current column visible (or the other columns invisible).</p>

Step	Procedure: Implementation in the Graphics Designer
2	Two <i>Windows Objects</i> → <i>Buttons</i> are assigned to each motor. These buttons control via <i>direct connections</i> the tags assigned to the individual motors.
3	To identify the currently selected motor, a <i>Standard Object</i> → <i>Rectangle</i> is assigned to each. At <i>Properties</i> → <i>Styles</i> → <i>Line Style</i> , the dotted line is selected and at <i>Properties</i> → <i>Styles</i> → <i>Fill Pattern</i> , the <i>Transparent</i> pattern. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i> , a <i>Dynamic Dialog</i> each is used to make the <i>Rectangle</i> visible only if the content of the <i>U08i_ex_tlg_00</i> tag agrees with the own object number.

C-Action at Motor1 (Group 1)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{

//show column 1
SetPropWord(lpszPictureName, "Control1", "Index", 0);
SetPropBOOL(lpszPictureName, "Control1", "ItemVisible", TRUE);

//hide column 2
SetPropWord(lpszPictureName, "Control1", "Index", 1);
SetPropBOOL(lpszPictureName, "Control1", "ItemVisible", FALSE);

//hide column 3
SetPropWord(lpszPictureName, "Control1", "Index", 2);
SetPropBOOL(lpszPictureName, "Control1", "ItemVisible", FALSE);

}
```

- Via the SetPropWord function, the Index 0 is set at the Control1 object. This corresponds to the first column. Via SetPropBOOL, this column is then set to visible.
- The same process is used to set the other columns to invisible.

Note for the General Application

The following adaptations must be made before the general application:

- The tags to be archived must be adapted to meet your own requirements.
- The graphical display of the objects must be adapted to meet your own requirements.

4.1.6 Archiving at Defined Times (ex_3_chapter_01e.pdf)

Task Definition

A cyclic-continuous process value archive is used to acquire process values in a cycle of one second. At every full minute, the sum of the values is to be archived.



The archived values are displayed in table form and the toolbar and status bar are to be implemented using standard tools.

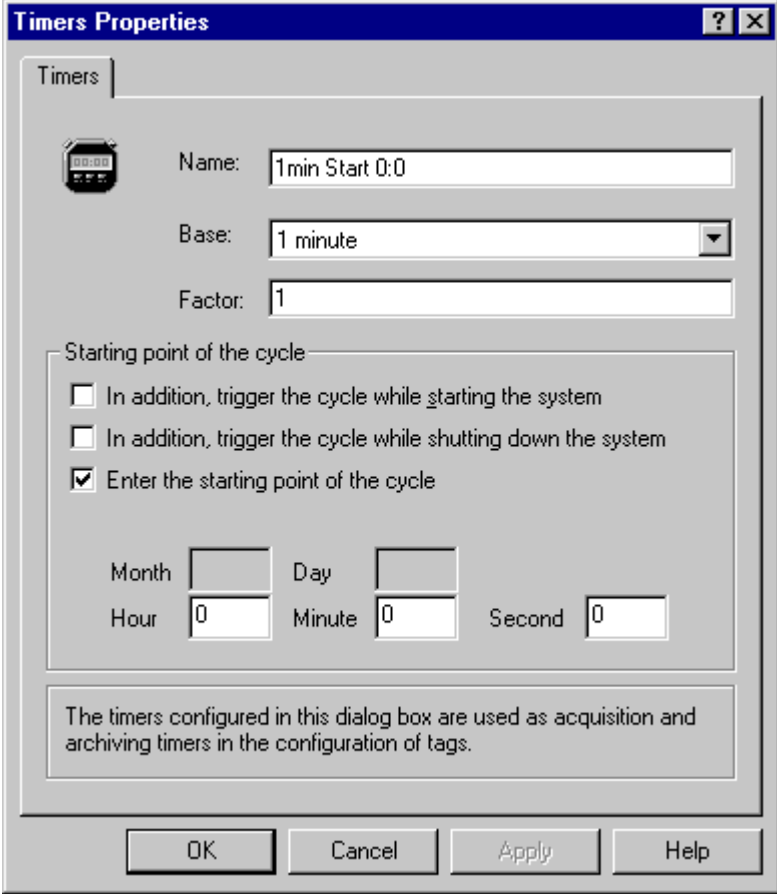
Implementation Concept

To archive the data to be displayed, a cyclic-continuous process value archive is created in the *Tag Logging* editor. To archive at every full minute, a new *Timer* is created. This timer is started at a defined time. The archiving is triggered by this timer.

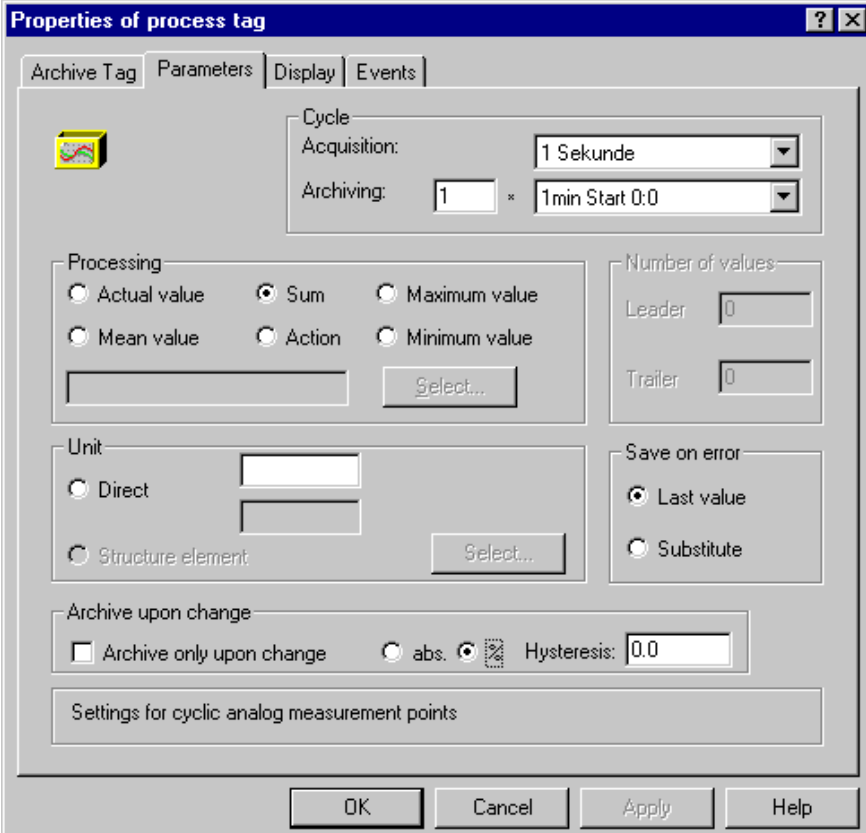
To display the data, a WinCC Online Table Control is created.

Creation of a New Timer

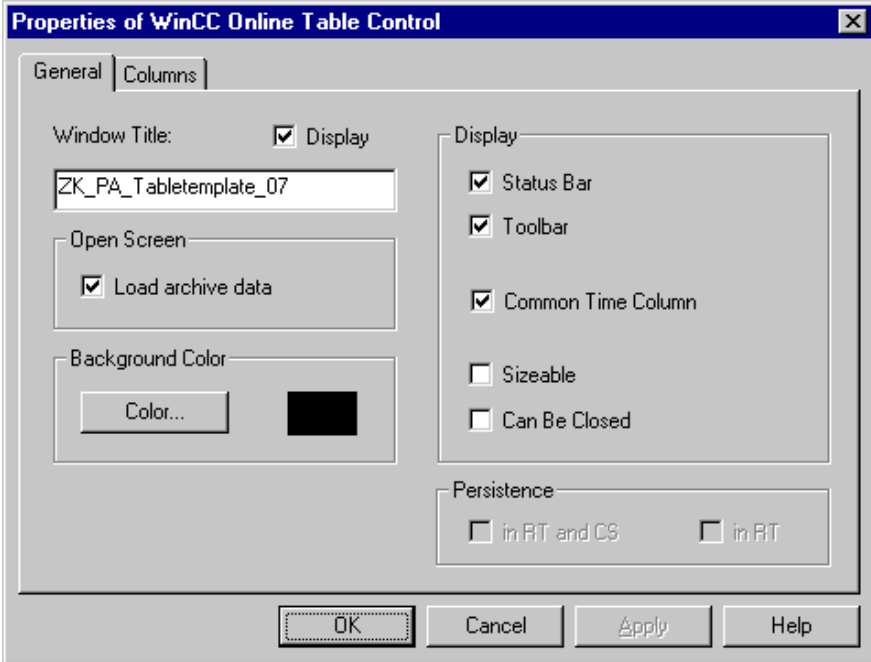
Step	Procedure: Creation of a New Timer
1	<p>Open the <i>Tag Logging</i> editor from the <i>WinCC Explorer</i>.</p> <p>Create a new <i>Timer</i> by  on the corresponding entry in the navigation window.</p> 

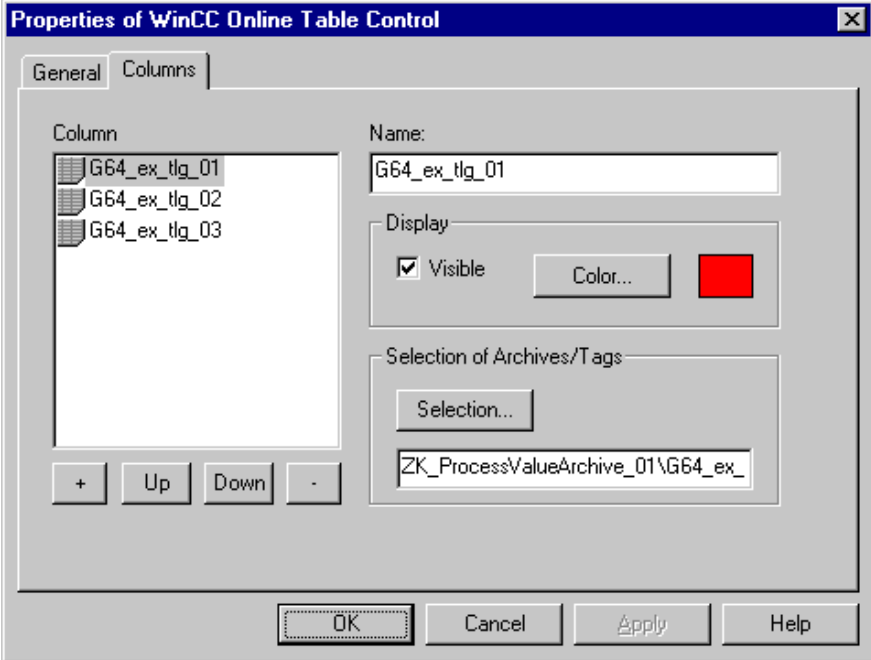
Step	Procedure: Creation of a New Timer																																				
2	<p>The properties dialog of the new <i>Timer</i> is displayed.</p> <p>As the <i>Name</i> of the <i>Timer</i>, this sample uses <i>1min Start 0:0</i>. As the <i>Base</i>, select <i>1 Minute</i> with a <i>Factor</i> of <i>1</i>. The <i>Factor</i> enables you to configure timers of, for example, four or six minutes. In the <i>Starting Point of the Cycle</i> field, select the <i>Enter the Starting Point of the Cycle</i> check-box. In each entry field of the cycle, <i>0</i> is entered. This causes the cycle to be triggered after the first full minute of runtime has elapsed. If a concrete time would be entered, the cycle would be started for the first time upon reaching the set time.</p> <p>Exit the dialog box by clicking on <i>OK</i>.</p> 																																				
3	<p>In the right window, the symbol of the newly created <i>timer</i> will be displayed in addition to the default timers.</p> <table border="1" data-bbox="527 1570 1263 1873"> <thead> <tr> <th>Timer name</th> <th>Time base</th> <th>Time factor</th> <th>Last change</th> </tr> </thead> <tbody> <tr> <td> 500 ms</td> <td>500 ms</td> <td>1</td> <td>10/22/97 01:14:28 PM</td> </tr> <tr> <td> 1 Sekunde</td> <td>1 second</td> <td>1</td> <td>10/22/97 01:14:28 PM</td> </tr> <tr> <td> 1 Minute</td> <td>1 minute</td> <td>1</td> <td>10/22/97 01:14:28 PM</td> </tr> <tr> <td> 1 Stunde</td> <td>1 hour</td> <td>1</td> <td>10/22/97 01:14:28 PM</td> </tr> <tr> <td> 1 Tag</td> <td>1 day</td> <td>1</td> <td>10/22/97 01:14:28 PM</td> </tr> <tr> <td> 1min Start 0:0</td> <td>1 minute</td> <td>1</td> <td>02/03/98 09:08:36 AM</td> </tr> <tr> <td> 1min Start 10:30</td> <td>1 minute</td> <td>1</td> <td>02/03/98 09:10:01 AM</td> </tr> <tr> <td> 1hour Start 0:0</td> <td>1 hour</td> <td>1</td> <td>02/04/98 03:59:48 PM</td> </tr> </tbody> </table>	Timer name	Time base	Time factor	Last change	500 ms	500 ms	1	10/22/97 01:14:28 PM	1 Sekunde	1 second	1	10/22/97 01:14:28 PM	1 Minute	1 minute	1	10/22/97 01:14:28 PM	1 Stunde	1 hour	1	10/22/97 01:14:28 PM	1 Tag	1 day	1	10/22/97 01:14:28 PM	1min Start 0:0	1 minute	1	02/03/98 09:08:36 AM	1min Start 10:30	1 minute	1	02/03/98 09:10:01 AM	1hour Start 0:0	1 hour	1	02/04/98 03:59:48 PM
Timer name	Time base	Time factor	Last change																																		
500 ms	500 ms	1	10/22/97 01:14:28 PM																																		
1 Sekunde	1 second	1	10/22/97 01:14:28 PM																																		
1 Minute	1 minute	1	10/22/97 01:14:28 PM																																		
1 Stunde	1 hour	1	10/22/97 01:14:28 PM																																		
1 Tag	1 day	1	10/22/97 01:14:28 PM																																		
1min Start 0:0	1 minute	1	02/03/98 09:08:36 AM																																		
1min Start 10:30	1 minute	1	02/03/98 09:10:01 AM																																		
1hour Start 0:0	1 hour	1	02/04/98 03:59:48 PM																																		



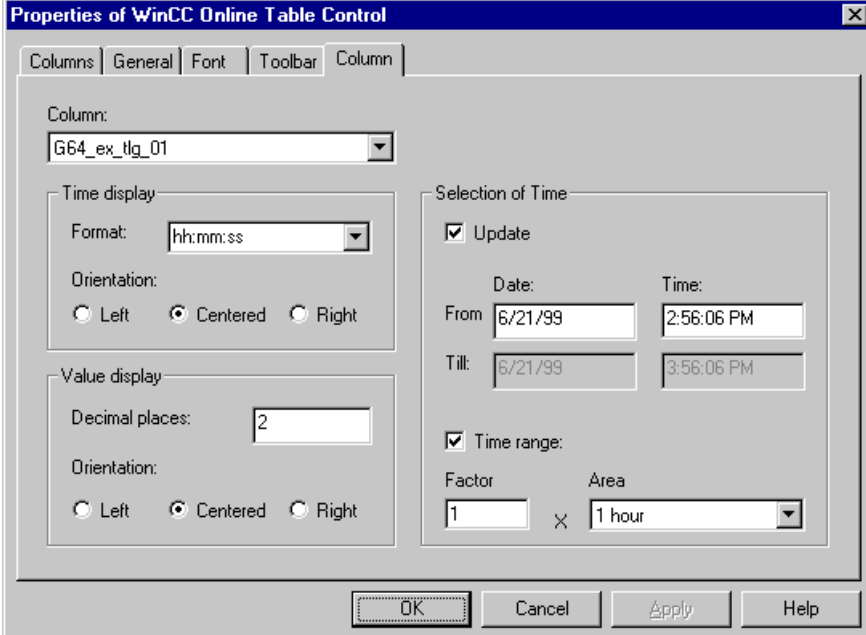
Creating a Process Value Archive

Step	Procedure: Creating a Process Value Archive
1	<p>Create a new <i>Process Value Archive</i> using the Archive Wizard. In this sample, the archive has been named <i>ZK_ProcessValueArchive_01</i>.</p> <p>As the tags to be archived, <i>G64_ex_tlg_01</i>, <i>G64_ex_tlg_02</i> and <i>G64_ex_tlg_03</i> have been selected. In this sample project, they are supplied with values by the simulator.</p> <p>In the properties dialog of the process value archive, the settings made by the Wizard are kept.</p>
2	<p>In the properties dialog of the first process tag, in the <i>Cycle</i> field of the <i>Parameters</i> tab, enter the cycle <i>1 Second</i> as the <i>Acquisition</i>. As the <i>Archiving</i>, set <i>1 * 1min Start 0:0</i>. As the <i>Processing</i>, the <i>Sum</i> radio-button is selected.</p> <p>For the remaining options of the first tag, the default settings are kept.</p> <p>For the remaining archive tags, the same settings are made as have just been performed.</p> 

Implementation in the Graphics Designer

Step	Procedure: Configuration of the Table Display
1	Create a new picture, in the sample this is the <i>ex_3_chapter_01e.pdl</i> picture.
2	Configuration of the Control used for displaying the table. This is the <i>WinCC Online Table Control</i> . It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture.
3	<p>After placing the Control in the picture, the <i>WinCC Online Table Control Properties</i> dialog is automatically opened.</p> <p>In the <i>General Information</i> tab, you can specify the Control's title and how it is labeled. As the window title, <i>ZK_ProcessValueArchive_01</i> is entered. Via the <i>Color</i> button, the <i>Background Color</i> of the table window is set to black.</p> <p>In the <i>Display</i> field, the <i>Sizeable</i> check-box is deselected and the <i>Shared Time Column</i> check-box is selected in this sample.</p> 

Step	Procedure: Configuration of the Table Display
4	<p>In the <i>Columns</i> tab, the columns to be displayed are specified in detail. For this sample, three columns are needed.</p> <p>One column has already been created. This column is renamed to <i>G64_ex_tlg_01</i>.</p> <p>Via the <i>Selection</i> button, the <i>Archive Tag</i> to be displayed can be assigned to the column. In this sample, the <i>Archive Tag G64_ex_tlg_01</i> of the previously created <i>ZK_ProcessValueArchive_01</i> archive is assigned to the column.</p> <p>Add two more columns. They are assigned the <i>BINi_ex_tlg_m2</i> and <i>BINi_ex_tlg_m3</i> tags, and their column names and colors are adapted.</p> 

Step	Procedure: Configuration of the Table Display
5	<p>Specific properties settings of the column. For this purpose, an expanded properties dialog is available. This dialog is opened via a  on the Control. The properties dialog described previously, on the other hand, is opened via a  on the Control while the CTRL key is pressed.</p> <p>The expanded properties dialog contains in addition to the already mentioned <i>General Information</i> and <i>Columns</i> tabs three additional tabs. In the <i>Column</i> tab, the following settings are made.</p> <p>The <i>Format</i> of the <i>Time Display</i> is set to <i>hh:mm:ss</i>. The <i>Orientation</i> of the <i>Time Display</i> and the <i>Value Display</i> is set to <i>Centered</i>. The <i>Time Range</i> is set to <i>1 x 1 Hour</i>.</p> <p>For the remaining options, the presettings are kept. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p> 

Note:

A brief description of the configurations performed in the *Graphics Designer* for the *ex_3_chapter_01e.PDL* picture can be found in the chapter *Bar Display* (*ex_3_chapter_01e*).

Note for the General Application

The following adaptations must be made before the general application:

- With the configurations described, it is possible to start archiving at certain times. In addition, values can be stored in an archive every full minute, hour, etc.
- The archiving cycle created in this sample must be adapted to meet your own requirements with regard to start time and the time basis used.

4.1.7 Exporting Archives (ex_3_chapter_01f.pdl)

Task Definition

A cyclic-continuous process value archive is to be exported as a CSV file once the maximum number of data records is reached. The archive is to be locked at system start and only be enabled after a button is pressed.

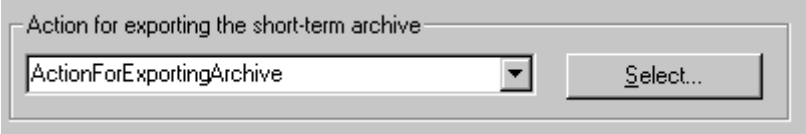
The archived values are displayed in table form and a user-defined toolbar and status bar are required. The user is to be informed about the time of the export via an information box.

Implementation Concept

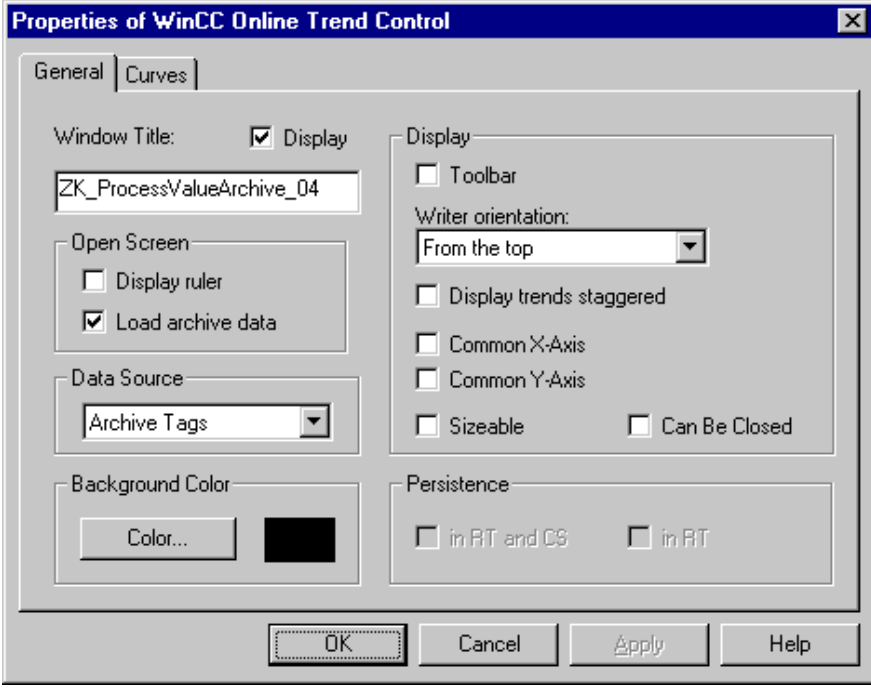
To archive the data to be displayed, a cyclic-continuous process value archive is created in the *Tag Logging* editor. For exporting the archive and to lock and release the archive, project functions are created.

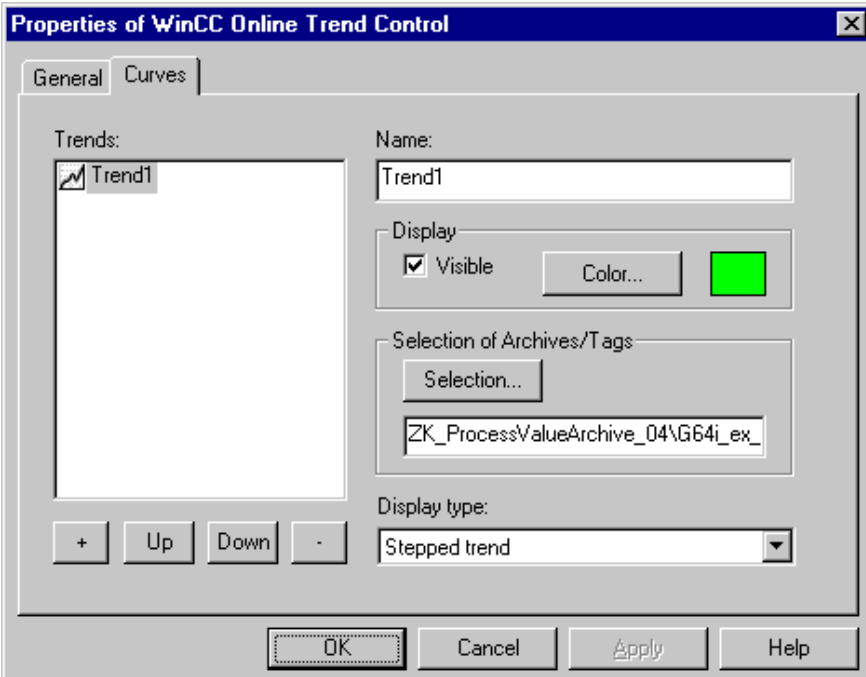
To display the data, a WinCC Online Trend Control is used. The toolbar consists of several *Windows Objects* → *Buttons* and *Smart Objects* → *Status Displays*.



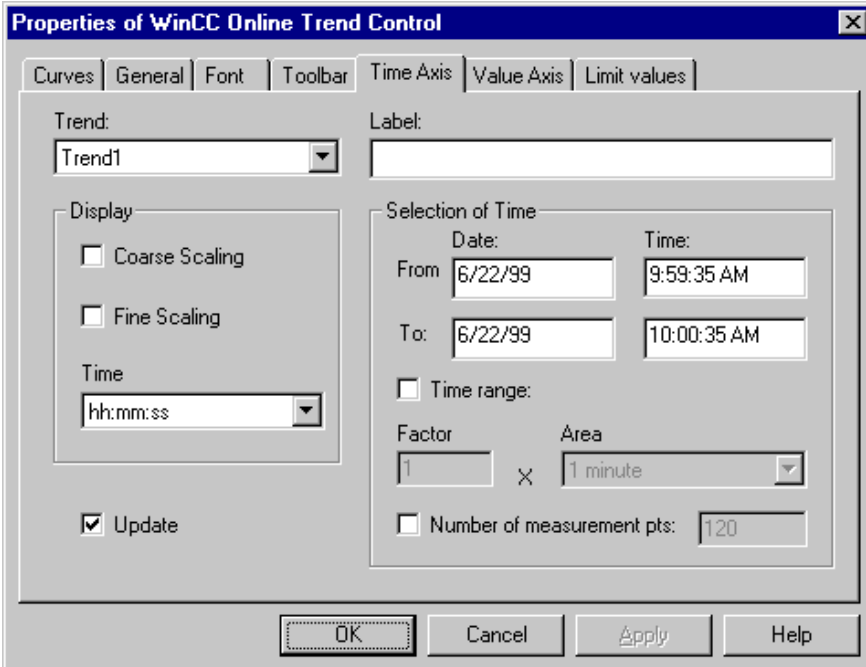
Creating a Process Value Archive

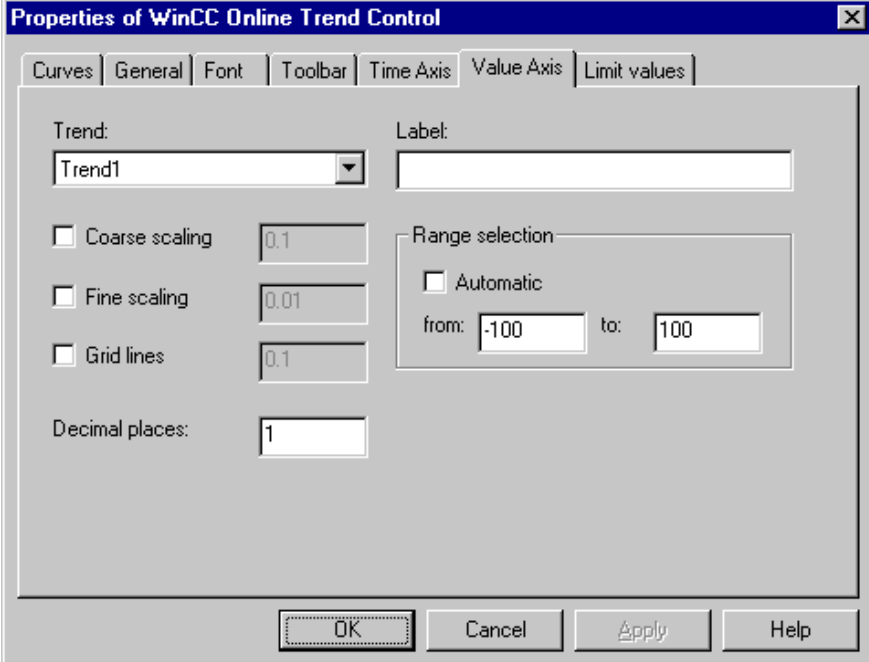
Step	Procedure: Creating a Process Value Archive
1	<p>Create a new <i>Process Value Archive</i> using the Archive Wizard. In this sample, the archive has been named <i>ZK_ProcessValueArchive_04</i>.</p> <p>As the tag to be archived, the <i>G64i_ex_tlg_04</i> tag is selected. In the sample project, this tag is supplied with the sum of the three trend profiles by the simulator.</p>
2	<p>In the properties dialog of the process value archive, the size of the archive is set to 200 data records. As the <i>Action for Exporting the Short-Term Archive</i>, the project function <i>ActionForExportingArchive</i> is set.</p> <p>For the remaining options, the default settings are kept.</p> 
3	<p>In the properties dialog of the process tag, the default settings are kept.</p>

Configuration of the Trend Display

Step	Procedure: Configuration of the Trend Display
1	Create a new picture, in the sample this is the <i>ex_3_chapter_01f.pdl</i> picture.
2	Configuration of the Control used for displaying the trend. This is the <i>WinCC Online Trend Control</i> . It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture.
3	<p>After placing the Control in the picture, its configuration dialog will be opened automatically.</p> <p>In the <i>General Information</i> tab, you can specify the Control's title and how it is labeled. As the window title, <i>ZK_ProcessValueArchive_04</i> is entered. Via the <i>Color</i> button, the <i>Background Color</i> of the trend window is set to black.</p> <p>In the <i>Display</i> field, all check-boxes are deselected and the text orientation is set to <i>from the top</i>.</p> 

Step	Procedure: Configuration of the Trend Display
4	<p>In the <i>Trends</i> tab, the trends to be displayed are specified in detail. For this sample, only one trend is needed.</p> <p>One trend has already been created. It is renamed to <i>Trend1</i>.</p> <p>Via the <i>Selection</i> button, the <i>Archive Tag</i> to be displayed can be assigned to the trend. In this sample, the <i>Archive Tag G64_ex_tlg_04</i> of the previously created <i>ZK_ProcessValueArchive_04</i> archive is assigned to the trend. The <i>Display Type</i> of the trend is set to <i>Stepped Trend</i>.</p> 

Step	Procedure: Configuration of the Trend Display
5	<p>Specific properties settings of the trend. For this purpose, an expanded properties dialog is available. This dialog is opened via a  on the Control. The properties dialog described previously, on the other hand, is opened via a  on the Control while the CTRL key is pressed.</p> <p>The expanded properties dialog contains in addition to the already mentioned <i>General Information</i> and <i>Trends</i> tabs five additional tabs.</p> <p>In the <i>Time Axis</i> trend, the following settings are made. The <i>Coarse Grid</i> and <i>Fine Grid</i> check-boxes are deselected and the <i>Time Format</i> is set to <i>hh:mm:ss</i>. In the <i>Time Selection</i> field, the <i>Time Range</i> check-box is deselected.</p> 

Step	Procedure: Configuration of the Trend Display
6	<p>In the <i>Value Axis</i> tab, all check-boxes are deselected from the <i>Trend</i> field. In the <i>Area Selection</i> field, the <i>Automatic</i> check-box is deselected and the values from <i>-100 to 100</i> are entered.</p> <p>For the remaining options, the presettings are kept. The properties dialog of the Control can be closed by clicking on <i>OK</i>.</p>  <p>The screenshot shows the 'Properties of WinCC Online Trend Control' dialog box with the 'Value Axis' tab selected. The 'Trend' dropdown menu is set to 'Trend1'. The 'Label' field is empty. Under the 'Range selection' section, the 'Automatic' checkbox is unchecked, and the 'from' field is set to '-100' and the 'to' field is set to '100'. Other options like 'Coarse scaling', 'Fine scaling', 'Grid lines', and 'Decimal places' are set to their default values (0.1, 0.01, 0.1, and 1 respectively). The 'OK' button is highlighted.</p>

Project Function for Exporting the Archive

```

void ActionForExportingArchive (LPTSTR lpszArchivNameReturn, LPTSTR lpszVar
{
    BOOL fRet;
    int iTlgCon = 0;
    CMN_ERROR Error;
    char szProj[MAX_PATH];
    char szFile[MAX_PATH];
    LPTSTR lpszArchivName =
        "PDE#HD#ZK_ProcessValueArchive_04#G64i_ex_tlg_04" ;
    char szFileName[MAX_PATH] = "";
    LPTSTR lpszFileName;
    TLG_IO_BACKUP_SELECT ibs;
    DWORD dwSize;
    time_t Time;
    struct tm* TimeStruct;
    int nPathLen, nFileLen;

    DMGetRuntimeProject( szProj, MAX_PATH, &Error);

    nPathLen=strlen(szProj);
    nFileLen=strlen((strrchr(szProj, '\\')+1));

    strcat(szFile, szProj, nPathLen-nFileLen);
    sprintf(szFileName, "%s%s", szFile, "ArchiveBackUp.CSV");
    lpszFileName=&szFileName[0];

    time(&Time);
    TimeStruct = localtime(&Time);

    ibs.sysFrom.wYear = 1997;
    ibs.sysFrom.wMonth = 1;
    ibs.sysFrom.wDay = 1;
    ibs.sysFrom.wHour = 0;
    ibs.sysFrom.wMinute = 0;
    ibs.sysFrom.wSecond = 0;

    ibs.sysTo.wYear = (WORD)(TimeStruct->tm_year+1900);
    ibs.sysTo.wMonth = (WORD)(TimeStruct->tm_mon+1);
    ibs.sysTo.wDay = (WORD)(TimeStruct->tm_mday);
    ibs.sysTo.wHour = (WORD)(TimeStruct->tm_hour);
    ibs.sysTo.wMinute = (WORD)(TimeStruct->tm_min);
    ibs.sysTo.wSecond = (WORD)(TimeStruct->tm_sec);

    fRet = TLGConnect( NULL, &Error );
    if (fRet==FALSE) printf("Error in TLGConect(...)\r\n");

    fRet=TLGGetBackupSize (lpszArchivName, &dwSize, &ibs, TLG_BACKUP_EVACUATE,
    if (fRet==FALSE) printf("Error in TLGGetBackupSize(...)[%s]\r\n", Error.szEr
    else SetTagWord("U16i_ex_tlg_00", (WORD)dwSize);



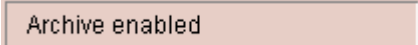
    fRet=TLGBackup (lpszArchivName, lpszFileName, &ibs, TLG_BACKUP_EXPORT, TLG_
    if (fRet==FALSE) printf("Error in TLGBackup(...) [%s]\r\n", Error.szErrorTex
    else SetTagBit("BINi_ex_tlg_09", TRUE);

    TLGDisconnect( NULL );
}

```

- Determination of the project path.
- Generation of the file name to which the archive is exported. This name also includes the path.
- Determination of the system time.
- Specification of the start and end time, between which the archiving times of the data to be exported are located.
- Establishing a connection to *Tag Logging* using the *TLGConnect* function.
- Determination of the size of the data to be exported using the *TLGBackupSize* function. This value is stored in an internal tag.
- Export of the archive using the *TLGBackup* function and setting of the binary tag *BINi_ex_tlg_09*, which makes the export visible.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	In Tag Management, create three tags of the <i>Binary Tag</i> type. In this sample, the <i>BINi_ex_tlg_06</i> , <i>BINi_ex_tlg_08</i> and <i>BINi_ex_tlg_09</i> tags are used. Additionally, the <i>U16i_ex_tlg_00</i> tag of the <i>Unsigned 16-Bit Value</i> type is required.
2	The implementation of a user-defined toolbar has already been described in detail in the Archiving if Values are Exceeded (ex_3_chapter_01b.pdl) sample. In this chapter, only the newly added control elements are described.
3	<p>To control the archiving, a <i>Windows Object</i> → <i>Button</i> is configured. In this sample, this is the <i>Button16</i> object.</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, a <i>C-Action</i> is created that inverts the status of the <i>BINi_ex_tlg_08</i> tag and calls the project function <i>LockUnlockArchive</i>. The binary tag is used to store the current status of the archive.</p> 
4	<p>Configure a new picture that will be displayed when the archive is exported. In the sample, this is the <i>ex_5_window_03.PDL</i> picture.</p> <p>This picture contains a <i>Standard Object</i> → <i>Static Text</i> which displays a text via a <i>C-Action</i>. This text consists of a fixed part and the number value of the <i>U16i_ex_tlg_00</i> tag. This tag contains the size of the exported data. Additionally, the picture contains a <i>Windows Object</i> → <i>Button</i> and a <i>Smart Object</i> → <i>Graphic Object</i> that both, via a <i>direct connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, switch the constant 0 to the <i>BINi_ex_tlg_09</i> tag.</p> 
5	In the initial picture, configure a <i>Smart Object</i> → <i>Picture Window</i> ; in the sample this is the <i>Picture Window1</i> object. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> , set the <i>ex_5_window_03.PDL</i> picture. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i> , create a <i>tag connection</i> to the <i>BINi_ex_tlg_09</i> tag.
6	<p>To display the status bar, two <i>Windows Objects</i> → <i>Buttons</i> are configured; in this sample, these are the <i>Button14</i> and <i>Button17</i> objects.</p> <p>For <i>Button17</i>, create a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Font</i> → <i>Text</i> that, dependent on the <i>BINi_ex_tlg_06</i> tag, either returns the text <i>Update Started</i> or <i>Update Stopped</i> to the property.</p> <p>For <i>Button14</i>, create a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Font</i> → <i>Text</i> that, dependent on the <i>BINi_ex_tlg_08</i> tag, either returns the text <i>Archive Enabled</i> or <i>Archive Locked</i> to the property.</p> 

Project Function for Locking and Enabling the Archive

```

void LockUnlockArchive (BOOL bLock)
{
    BOOL fRet;
    CMN_ERROR Error;
    LPTSTR      lpszArchivName = "ZK_ProcessValueArchive_04";

    fRet = TLGConnect( NULL, &Error );

    if (fRet==FALSE)
        printf("Error in TLGConnect(...)\r\n");
    else
    {
        fRet=TLGLockArchiv (NULL,lpszArchivName,bLock,&Error );
        if (fRet==FALSE) printf("Error in TLGLockArchive(...) [%s]\r\n",
            Error.szErrorText);
        TLGDisconnect( NULL );
    }
}

```

- Establishing a connection to *Tag Logging*.
- Call of the *TLGLockArchive* function. The transfer parameter *bLock* decides whether the archive is locked or enabled.


Note for the General Application

The following adaptations must be made before the general application:

- The tags to be archived must be adapted to meet your own requirements.
- The maximum archive size and the path and file name of the export file must be adapted.

4.2 Alarm Logging



In runtime, the samples pertaining to this topic are accessed by selecting the button displayed above using the . The samples are configured in the *ex_3_chapter_02.pdl* to *ex_3_chapter_02d.pdl* pictures.

General Information

The *Alarm Logging* editor is responsible for message acquisition and archiving. The editor contains functions for transferring messages from processes, for processing messages, for displaying messages, for acknowledging messages and for archiving messages.

In this way, Alarm Logging supports the user with locating error causes.

4.2.1 Bit Message Procedure (ex_3_chapter_02.pdl)

Task Definition

Four motors are to be monitored by *Alarm Logging*. Errors are displayed by setting various bits within a tag assigned to each motor. The message stati of the motors are stored in internal tags. Dependent on the message status, the display of the motor is to be changed. The messages are to be displayed in a message window.

Implementation Concept

In *Alarm Logging*, several individual messages must be created that refer to the four motors monitored.

The message window is implemented in the Graphics Designer using a WinCC Alarm Control. The individual motors are displayed using several *Standard Objects*. The display changes of the motors at the different message stati are realized using *C-Actions*.

Creation of the Required Tags


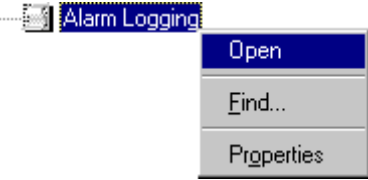

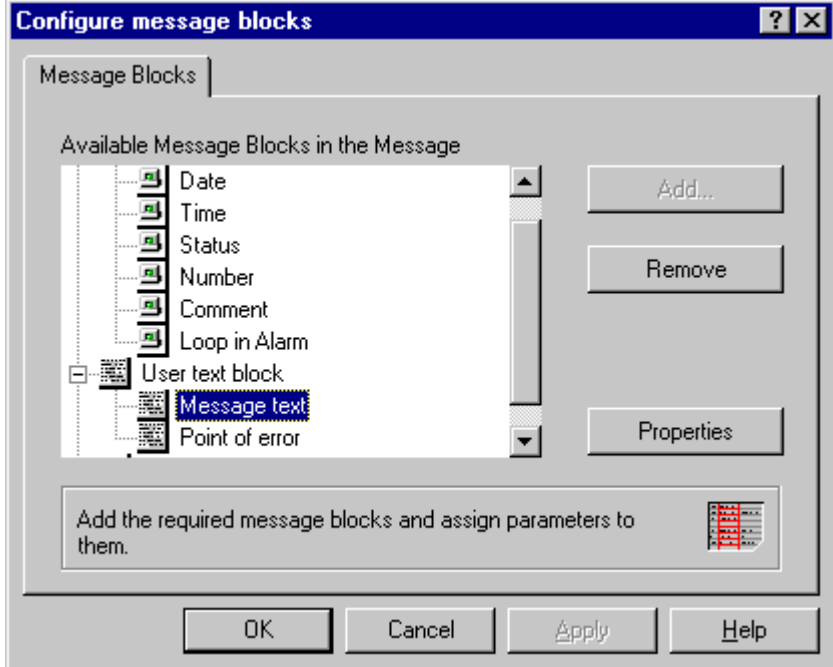


Step	Procedure: Creation of the Required Tags
1	<p>Creation of a total of twelve tags of the <i>Unsigned 16-Bit Value</i> type in Tag Management.</p> <p>Four of these tags are used as event tags. In the sample, these are the <i>U16i_ex_alg_00</i>, <i>U16i_ex_alg_03</i>, <i>U16i_ex_alg_06</i> and <i>U16i_ex_alg_09</i> tags. Four other tags are used as status tags; in the sample, these are the <i>U16i_ex_alg_02</i>, <i>U16i_ex_alg_05</i>, <i>U16i_ex_alg_08</i> and <i>U16i_ex_alg_11</i> tags. The remaining tags, <i>U16i_ex_alg_12</i>, <i>U16i_ex_alg_13</i>, <i>U16i_ex_alg_14</i> and <i>U16i_ex_alg_15</i> in the sample, serve as acknowledgment tags.</p>

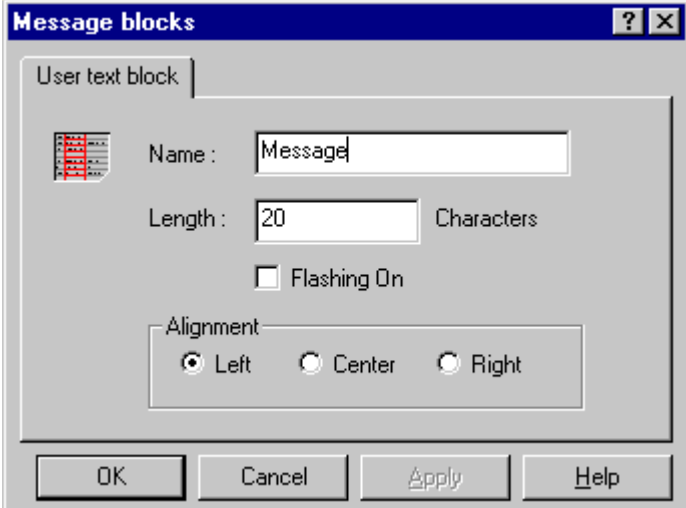
Message Blocks

A message consists of various message blocks. They can be categorized into three areas:




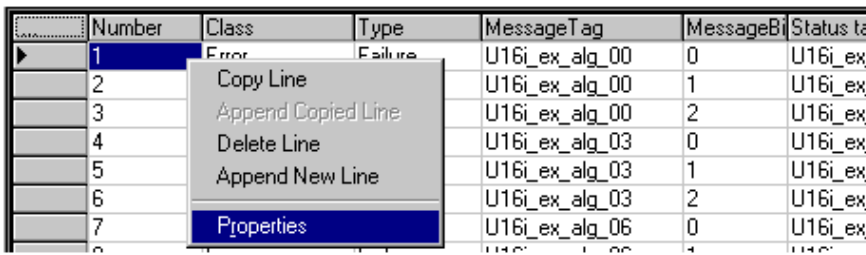
- **System Blocks:** They contain system data that is assigned by *Alarm Logging*. This includes the date, time, reporting identification, etc.
- **Process Value Blocks:** They contain the values returned by the process, e.g. critical fill levels, temperatures, etc.
- **User Text Blocks:** Texts that contribute to the general information and understanding, e.g. error explanations, message causes, error locations, etc.

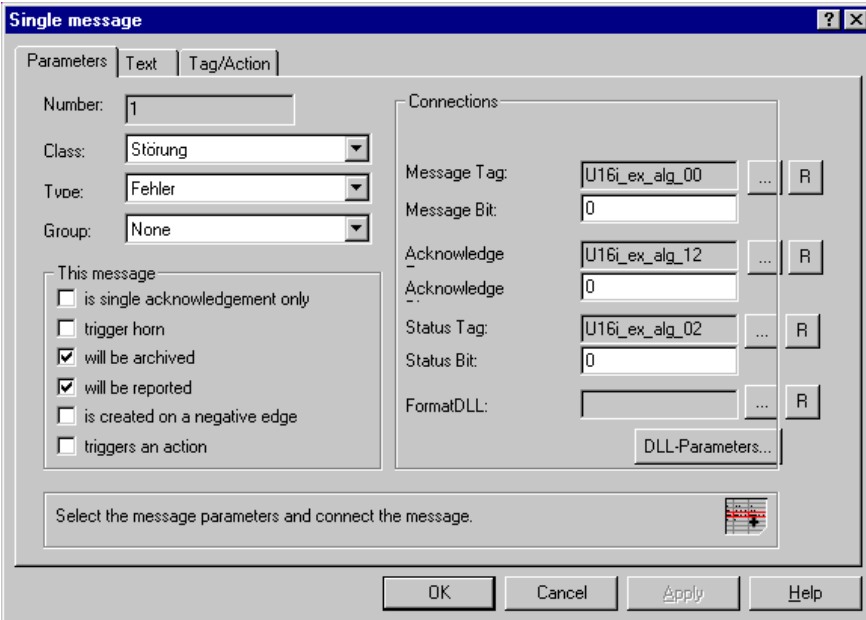
Message Block Configuration

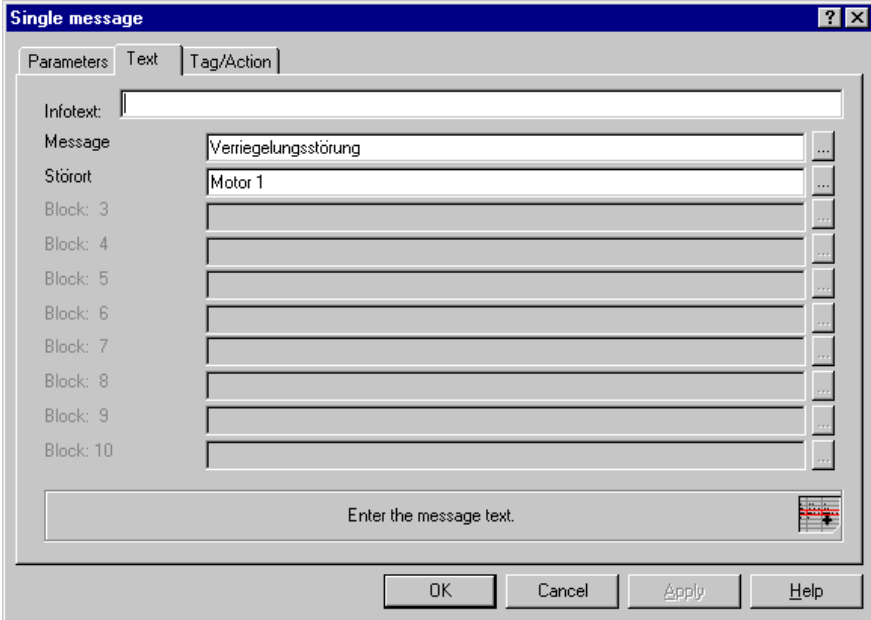
Step	Procedure: Message Block Configuration
1	<p>In the <i>WinCC Explorer</i>, open the <i>Alarm Logging</i> editor by  on it and then selecting <i>Open</i> from the pop-up menu.</p> 
2	<p>Selection of the desired message blocks. This is done via a  on the <i>Message Blocks</i> entry and then selecting <i>Message Blocks</i> from the pop-up menu. The <i>Configure Message Blocks</i> dialog is opened.</p> 
3	<p>Via the <i>Add</i> button, the dialog for adding blocks is opened for the selected system blocks entry, user text blocks entry or process value blocks entry.</p> <p>If a block is selected with the  in the <i>Configure Message Blocks</i> dialog, the buttons <i>Remove</i> and <i>Properties</i> become operational.</p> <p>The first button allows you to remove selected blocks, the second button allows you to configure the properties of the individual message blocks.</p> 

Step	Procedure: Message Block Configuration
4	<p>In this sample, the <i>Name</i> is kept and the <i>Length</i> is set to 20 characters. Clicking on <i>OK</i> finalizes the settings made in the <i>Message Blocks</i> dialog. The <i>Configure Message Blocks</i> dialog can also be closed by clicking on <i>OK</i>.</p> 






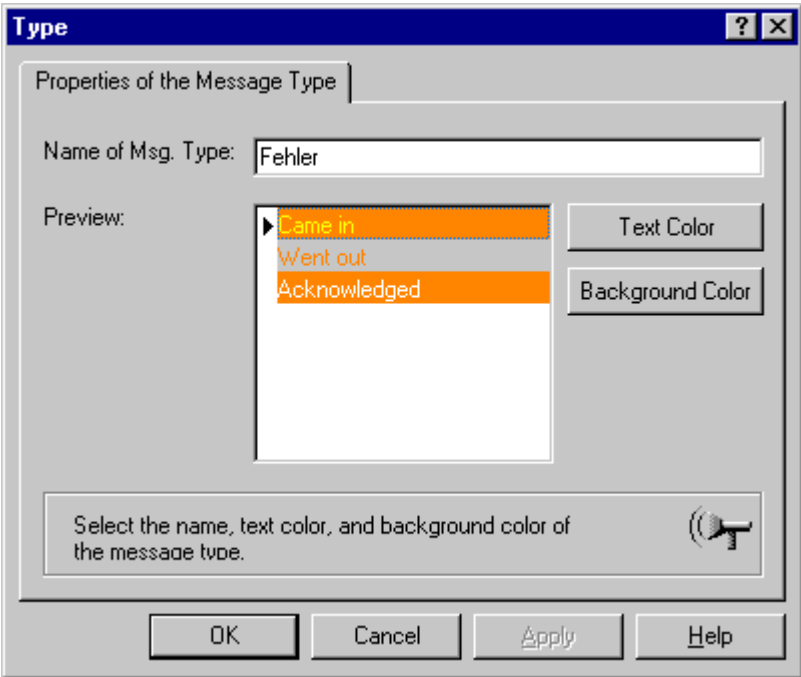
Creation of Single Messages

Step	Procedure: Creation of Single Messages
1	<p>In the <i>Alarm Logging</i> editor, a table window is located in the lower area. In this area, the single messages are configured and already configured ones displayed. Via a , new lines can be added.</p>  <p>For this sample, a total of 12 different single messages are created. Each message corresponds to one line in the table window and consists of a number of columns. Settings can be made directly in the individual columns. In this sample, however, the settings are made via the <i>Single Message</i> dialog. This dialog is opened via a  on the appropriate message line.</p> 


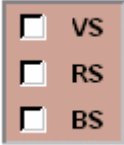
Step	Procedure: Message Block Configuration
2	<p>Open the <i>Single Message</i> dialog for the first line as described in the previous step.</p> <p>In the <i>Parameters</i> tab, select the message class <i>Error</i> and the message type <i>Failure</i>.</p> <p>In the <i>This Message</i> field, select the check-boxes <i>will be archived</i> and <i>will be reported</i>.</p> <p>In the <i>Connections</i> field, select as the <i>Event Tag</i> the tag <i>U16i_ex_alg_00</i>. As the <i>Event Bit</i>, <i>0</i> is entered. This means that the message will be generated if the first bit of the tag set takes on the status 1.</p> <p>As the <i>Acknowledge Tag</i>, the tag <i>U16i_ex_alg_12</i> is selected and as the <i>Acknowledge Bit</i>, <i>0</i> is entered. In other words, if the message is acknowledged in runtime, the first bit of the tag set is set to 1.</p> <p>As the <i>Status Tag</i>, the tag <i>U16i_ex_alg_02</i> is selected and as the <i>Status Bit</i>, <i>0</i> is entered. This setting means that the first bit of the tag set represents the <i>Came in/Went out Status</i> of the message. If the message is pending, this bit is set to 1, if the message is no longer pending, this bit is reset. The ninth bit of the tag contains the <i>Acknowledge Status</i> of the message. If it is not acknowledged, the bit has the status 1, if it is acknowledged, the status 0.</p> <p>A status tag of 16 bits can represent the stati of 8 single messages. The Low Byte contains the <i>Came in/Went out Stati</i> and the High-Byte the <i>Acknowledge Stati</i>.</p> 

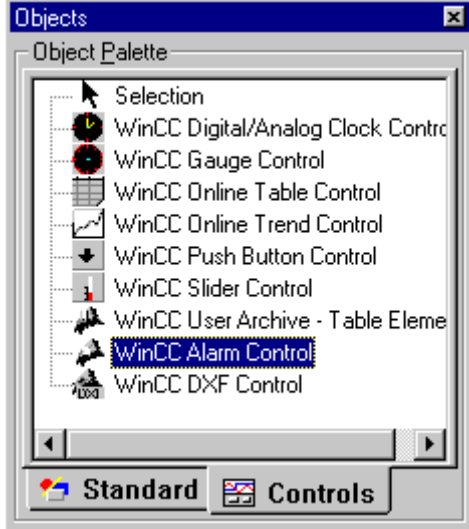

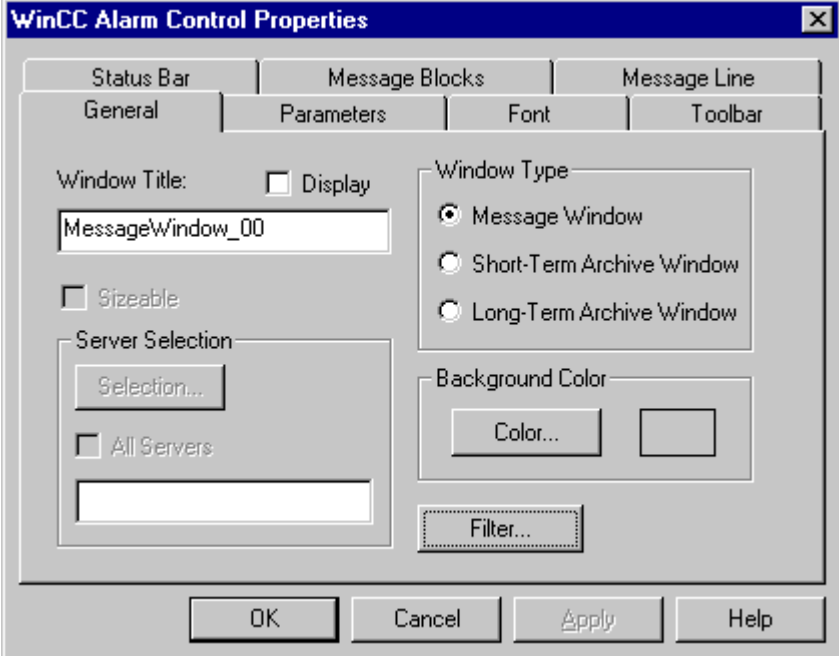
Step	Procedure: Message Block Configuration																																																																																																																					
3	<p>In the <i>Text</i> tab, the <i>Message Text Locking Error</i> and the <i>Point of Error Motor 1</i> are entered. No <i>Infotext</i> is used. The <i>Tag/Action</i> tab must not be filled out for this sample.</p> <p>The settings made are applied by clicking on <i>OK</i>.</p> 																																																																																																																					
4	<p>The message just created monitors the first of the four motors. For the first motor, two more message lines are created.</p> <p>The settings are made as described in step 2 to 3, but the <i>Event Bits</i>, <i>Acknowledge Bits</i> and <i>Status Bits</i> are adapted. Also, the <i>Message Texts Feedback Error</i> and <i>Bimetal Error</i> are used.</p>																																																																																																																					
5	<p>For the other three motors, three message lines each are also created.</p> <p>Here, the <i>Event Tags</i>, <i>Acknowledge Tags</i> and <i>Status Tags</i> as well as the texts for the <i>Point of Error</i> must be adapted to each motor.</p> <table border="1" data-bbox="527 1339 1393 1591"> <thead> <tr> <th>...</th> <th>Number</th> <th>Class</th> <th>Type</th> <th>MessageTag</th> <th>MessageB</th> <th>Status tag</th> <th>Status bit</th> <th>Message text</th> </tr> </thead> <tbody> <tr> <td></td> <td>1</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_00</td> <td>0</td> <td>U16i_ex_alg_02</td> <td>0</td> <td>Lock Error</td> </tr> <tr> <td></td> <td>2</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_00</td> <td>1</td> <td>U16i_ex_alg_02</td> <td>1</td> <td>Feedback Error</td> </tr> <tr> <td></td> <td>3</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_00</td> <td>2</td> <td>U16i_ex_alg_02</td> <td>2</td> <td>Bimetal Error</td> </tr> <tr> <td></td> <td>4</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_03</td> <td>0</td> <td>U16i_ex_alg_05</td> <td>0</td> <td>Lock Error</td> </tr> <tr> <td></td> <td>5</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_03</td> <td>1</td> <td>U16i_ex_alg_05</td> <td>1</td> <td>Feedback Error</td> </tr> <tr> <td></td> <td>6</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_03</td> <td>2</td> <td>U16i_ex_alg_05</td> <td>2</td> <td>Bimetal Error</td> </tr> <tr> <td></td> <td>7</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_06</td> <td>0</td> <td>U16i_ex_alg_08</td> <td>0</td> <td>Lock Error</td> </tr> <tr> <td></td> <td>8</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_06</td> <td>1</td> <td>U16i_ex_alg_08</td> <td>1</td> <td>Feedback Error</td> </tr> <tr> <td></td> <td>9</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_06</td> <td>2</td> <td>U16i_ex_alg_08</td> <td>2</td> <td>Bimetal Error</td> </tr> <tr> <td></td> <td>10</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_09</td> <td>0</td> <td>U16i_ex_alg_11</td> <td>0</td> <td>Lock Error</td> </tr> <tr> <td></td> <td>11</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_09</td> <td>1</td> <td>U16i_ex_alg_11</td> <td>1</td> <td>Feedback Error</td> </tr> <tr> <td></td> <td>12</td> <td>Error</td> <td>Failure</td> <td>U16i_ex_alg_09</td> <td>2</td> <td>U16i_ex_alg_11</td> <td>2</td> <td>Bimetal Error</td> </tr> </tbody> </table>	...	Number	Class	Type	MessageTag	MessageB	Status tag	Status bit	Message text		1	Error	Failure	U16i_ex_alg_00	0	U16i_ex_alg_02	0	Lock Error		2	Error	Failure	U16i_ex_alg_00	1	U16i_ex_alg_02	1	Feedback Error		3	Error	Failure	U16i_ex_alg_00	2	U16i_ex_alg_02	2	Bimetal Error		4	Error	Failure	U16i_ex_alg_03	0	U16i_ex_alg_05	0	Lock Error		5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error		6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error		7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error		8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error		9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error		10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error		11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error		12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error
...	Number	Class	Type	MessageTag	MessageB	Status tag	Status bit	Message text																																																																																																														
	1	Error	Failure	U16i_ex_alg_00	0	U16i_ex_alg_02	0	Lock Error																																																																																																														
	2	Error	Failure	U16i_ex_alg_00	1	U16i_ex_alg_02	1	Feedback Error																																																																																																														
	3	Error	Failure	U16i_ex_alg_00	2	U16i_ex_alg_02	2	Bimetal Error																																																																																																														
	4	Error	Failure	U16i_ex_alg_03	0	U16i_ex_alg_05	0	Lock Error																																																																																																														
	5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error																																																																																																														
	6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error																																																																																																														
	7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error																																																																																																														
	8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error																																																																																																														
	9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error																																																																																																														
	10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error																																																																																																														
	11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error																																																																																																														
	12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error																																																																																																														



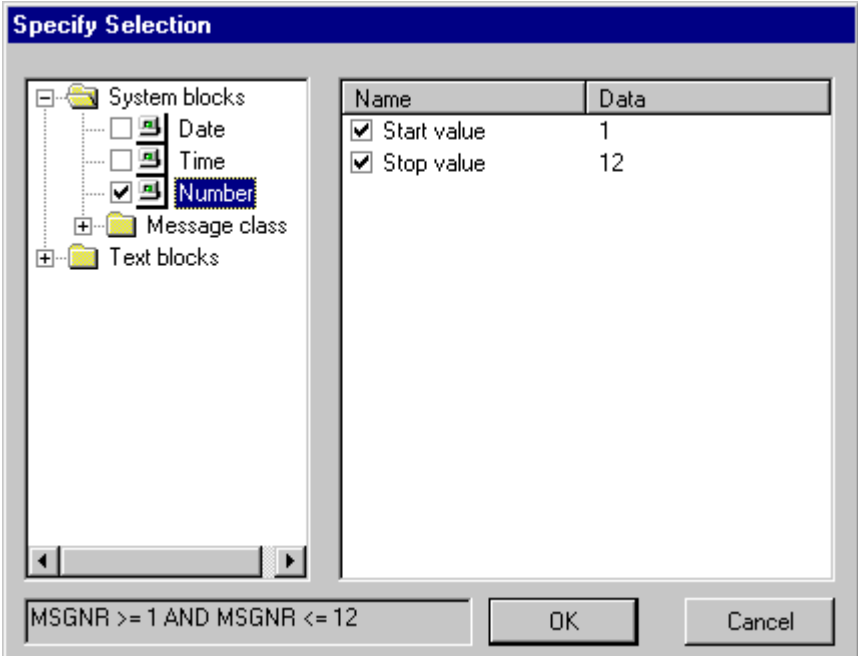
Color Scheme Configuration of the Messages

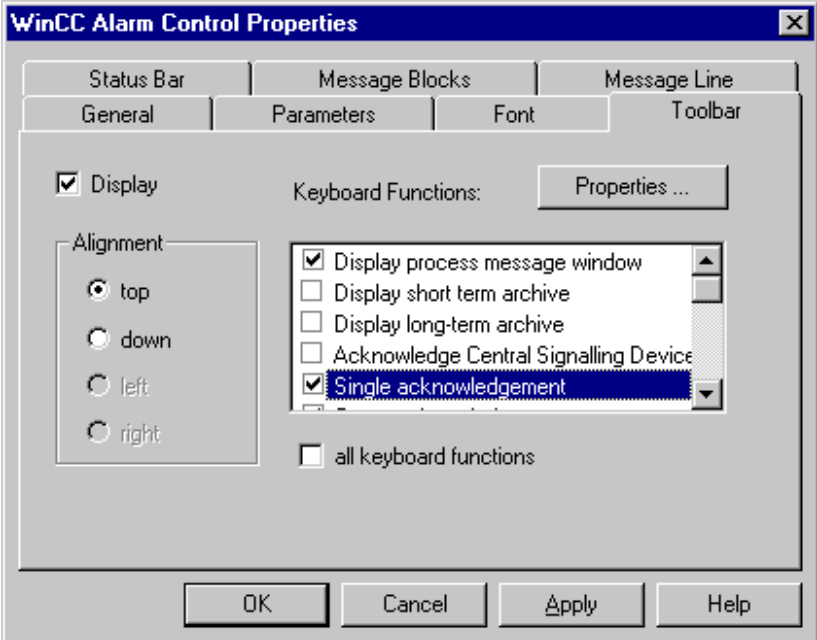

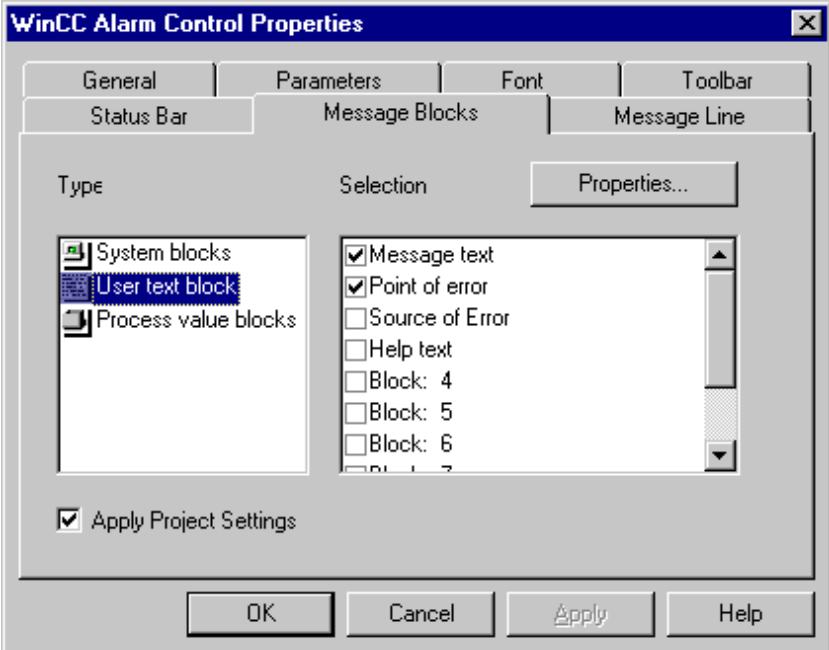
Step	Procedure: Color Scheme Configuration of the Messages
1	<p>The configured single messages are of the message class <i>Error</i> and the message type <i>Failure</i>.</p> <p>Via a  on the <i>Message Classes</i> entry, all available message classes will be displayed in the right window. Via a  on the icon of the message class <i>Error</i>, all message types pertaining to this class will be displayed. Via a  on the icon of the message type <i>Failure</i> or via a  on it and then selecting properties from the pop-up menu, the <i>Type</i> dialog is opened.</p> 
2	<p>In the <i>Type</i> dialog, a color scheme can be created for each message status.</p> <p>In this sample, the following color scheme is used:</p> <p>Came in: Text = Yellow, Background = Orange</p> <p>Went out: Text = Orange, Background = Light Gray</p> <p>Acknowledged: Text = White, Background = Orange</p> 
3	<p>The configurations made in <i>Alarm Logging</i> are saved via the <i>File</i> → <i>Save</i> menus.</p>


Implementation in the Graphics Designer


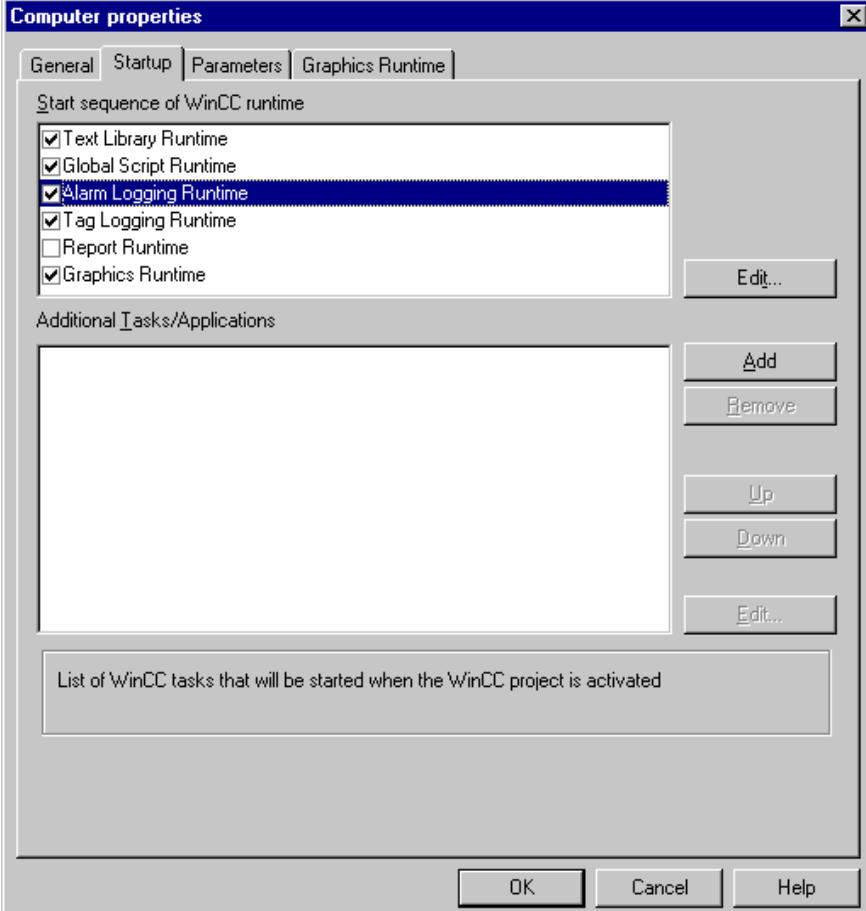
Step	Procedure: Implementation in the Graphics Designer
1	Creation of a new picture; in this project, this is the <i>ex_3_chapter_02</i> picture.
2	<p>The individual motors are displayed via a <i>Standard Object</i> → <i>Circle</i>, a <i>Standard Object</i> → <i>Static Text</i> and a <i>Standard Object</i> → <i>Polygon</i>.</p> <p>The motor is to change its color scheme when an error occurs or this message is acknowledged. This color scheme is to correspond to the message stati came in, went out and acknowledged.</p> <p>For this, a <i>C-Action</i> is created for the <i>Static Text</i> at <i>Properties</i> → <i>Colors</i> → <i>Font Color</i>, which changes the font color dependent on the current status of the status tag belonging to the motor.</p> <p>Likewise, a <i>C-Action</i> is created for the <i>Circle</i> at <i>Properties</i> → <i>Colors</i> → <i>Background Color</i> that performs the same task.</p> 
3	<p>The occurrence of an error at a motor is simulated via a <i>Windows Object</i> → <i>Check-Box</i>.</p> <p>At <i>Properties</i> → <i>Geometry</i> → <i>Number of Boxes</i>, 3 is entered.</p> <p>At <i>Properties</i> → <i>Output/Input</i> → <i>Selected Boxes</i>, a <i>Tag Connection</i> to the motor's corresponding event tag is created.</p> 

Step	Procedure: Implementation in the Graphics Designer
4	<p>To display the messages configured in <i>Alarm Logging</i>, a <i>WinCC Alarm Control</i> is used. It is selected from the Object Palette's <i>Control</i> selection menu and then placed in the picture.</p> 
5	<p>After placing the Control in the picture, its <i>configuration dialog</i> will be displayed automatically. This dialog can be closed by clicking on <i>OK</i>.</p> <p>Open the Control's <i>Properties</i> dialog. This dialog is displayed via a  on the Control. In the <i>General Information</i> tab, the <i>Selection</i> button is used to select the single messages created in <i>Alarm Logging</i> to be displayed by the Control.</p> 

Step	Procedure: Implementation in the Graphics Designer						
6	<p>Via a  on the system block <i>Number</i>, 2 check-boxes will be displayed in the right window. The <i>Start Value</i> is changed via a  on the default value 0 to 1 and the <i>Stop Value</i> to 12. This means that the Control only displays the single messages from number 1 to 12.</p>  <p>Specify Selection</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/> Start value</td> <td>1</td> </tr> <tr> <td><input checked="" type="checkbox"/> Stop value</td> <td>12</td> </tr> </tbody> </table> <p>MSGNR >= 1 AND MSGNR <= 12 OK Cancel</p>	Name	Data	<input checked="" type="checkbox"/> Start value	1	<input checked="" type="checkbox"/> Stop value	12
Name	Data						
<input checked="" type="checkbox"/> Start value	1						
<input checked="" type="checkbox"/> Stop value	12						

Step	Procedure: Implementation in the Graphics Designer
7	<p>In the <i>Toolbar</i> tab, the buttons to be displayed in runtime are selected. In this sample, the following buttons are required:</p> <p><i>Single Acknowledgment, Group Acknowledgment, Auto Scroll On/Off, Beginning of the List, End of the List, Next Message and Previous Message.</i></p> 
8	<p>In the <i>Message Blocks</i> tab, the columns are selected that will later be displayed in the message line. In this sample, system blocks are selected in the <i>Type</i> field using the . In the right window, <i>Date, Time and Number</i> are selected. For the User Text Blocks entry, <i>Message Text</i> and <i>Point of Error</i> are selected.</p> 

Step	Procedure: Implementation in the Graphics Designer
9	<p>In the <i>Message Line</i> tab, the previously selected <i>Message Blocks</i> are assigned to the message line. The <i>Available Message Blocks</i> fields lists the available columns. By pressing on the -> button, each message block can individually be added to the message line. Via the >> button, all message blocks listed in the window can be assigned to the message line at one time. The properties dialog is exited by clicking on <i>OK</i>.</p> 

Step	Procedure: Implementation in the Graphics Designer
10	<p>Activation of <i>Alarm Logging Runtime</i>.</p> <p>For this,  on the <i>Computer</i> entry in the <i>WinCC Explorer</i> and select <i>Properties</i> from the pop-up menu to open the <i>Computer List Properties</i> dialog. Click on the <i>Properties</i> button to open the properties dialog of the local computer.</p> <p>In the <i>Startup</i> tab, the applications to be activated with runtime are selected. The <i>Alarm Logging Runtime</i> check-box must be selected.</p> <p>The <i>Computer Properties</i> and <i>Computer List Properties</i> dialogs can be closed by clicking on <i>OK</i>.</p>  <p>The screenshot shows the 'Computer properties' dialog box with the 'Startup' tab selected. Under 'Start sequence of WinCC runtime', the following items are listed with checkboxes: Text Library Runtime (checked), Global Script Runtime (checked), Alarm Logging Runtime (checked and highlighted), Tag Logging Runtime (checked), Report Runtime (unchecked), and Graphics Runtime (checked). To the right of this list is an 'Edit...' button. Below this is the 'Additional Tasks/Applications' section, which is currently empty. To its right are buttons for 'Add', 'Remove', 'Up', 'Down', and 'Edit...'. At the bottom of the dialog is a text box labeled 'List of WinCC tasks that will be started when the WinCC project is activated' and three buttons: 'OK', 'Cancel', and 'Help'.</p>

C-Action at the Circle (Circle1)

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    DWORD state;

    state=GetTagDWord ("U16i_ex_alg_02");

    if ((state&1)||((state&2)||((state&4)))
        return 0x80FF;
    else
        return 0xFFFFF;

}
```

- This *C-Action* makes the Background Color property of the *Circle* assigned to the first motor dynamic.
- The status tag *U16i_ex_alg_02* assigned to the first motor is read. The Low Byte of this tag contains the message stati came in/went out, i.e. if the first, second or third bit of this tag is set to 1, the message is pending and the background color of the circle is set to orange (hex 80ff). If the message goes out, the background color is set to white (hex fffff).
- This *C-Action* is triggered upon the change of the status tag *U16i_ex_alg_02*.

C-Action at the Static Text (StaticText1)

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    DWORD state;

    State=GetTagDWord ("U16i_ex_alg_02");

    if ( ((state&1)&&(state&256)) || ((state&2)&&(state&512)) ||
        ((state&4)&&(state&1024)) )
        return 0xFFFF;
    else if ((state&1)||((state&2)||((state&4)))
        return 0xFFFFF;
    else if ((state&256)||((state&512)||((state&1024)))
        return 0x80FF;
    else
        return 0x800000;

}
```

- This *C-Action* makes the Font Color property of the *Static Text* assigned to the first motor dynamic.
- The status tag *U16i_ex_alg_02* assigned to the first motor is read. The Low Byte of this tag contains the message stati came in/went out, the High Byte the message status acknowledged. In the case of an unacknowledged, pending message message, the font color is set to yellow (hex ffff); in the case of an acknowledged message, the font color is set to white (hex fffff); in the case of an unacknowledged but gone out message, the font color is set to orange (hex 80ff). In the normal case, the font color is dark blue (hex 800000).
- This *C-Action* is triggered upon the change of the status tag *U16i_ex_alg_02*.

Note for the General Application

The following adaptations must be made before the general application:

- The required message blocks must be adapted to meet your own requirements.
- The event, status and acknowledge tags as well as their bits must be adapted to meet your own requirements.

4.2.2 Limit Value Monitoring (ex_3_chapter_02a.pdl)

Task Definition

The pressure and temperature values in three containers are to be monitored by *Alarm Logging*. If the analog values to be monitored come close to the critical range, warnings are to be generated. If they reach the critical range, alarms are generated. The occurrence of an alarm is to be reported optically and acoustically in the *Graphics Designer* as well. A largely user-defined message window layout is used.

Implementation Concept

In *Alarm Logging*, several individual messages must be created that refer to the three containers monitored.
The message window is created in the *Graphics Designer* using a WinCC Alarm Control.
The toolbar consists of several *Windows Objects* → *Buttons* and *Smart Objects* → *Status Displays*.



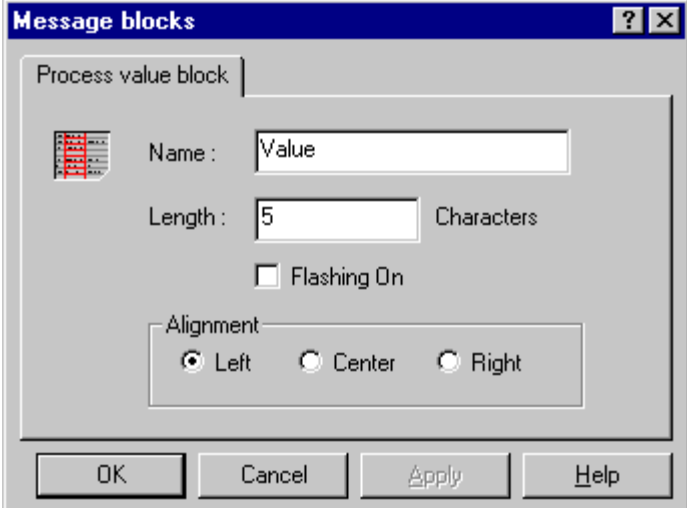
Creation of the Required Tags

Step	Procedure: Creation of the Required Tags
1	<p>Creation of a total of six tags of the <i>Unsigned 16-Bit Value</i> type in Tag Management. Three of these tags contain the temperature values of the individual containers. In the sample, these are the <i>U16i_ex_alg_t1</i>, <i>U16i_ex_alg_t2</i> and <i>U16i_ex_alg_t3</i> tags. The remaining three tags each contain the pressure values. In the sample, these are the <i>U16i_ex_alg_p1</i>, <i>U16i_ex_alg_p2</i> and <i>U16i_ex_alg_p3</i> tags.</p> <p>Three additional tags of the <i>Unsigned 16-Bit Value</i> type are required that are used as status tags. In the sample, these are the <i>U16i_ex_alg_01</i>, <i>U16i_ex_alg_04</i> and <i>U16i_ex_alg_07</i> tags.</p> <p>One tag of the <i>Unsigned 16-Bit Value</i> type for the control of the central indicator is required; in the sample, this is the <i>U16i_ex_alg_10</i> tag.</p> <p>Additionally, two tags of the <i>Binary Tag</i> type are required. In this sample, these are the <i>BINi_ex_alg_00</i> and <i>BINi_ex_alg_03</i> tags.</p>

Note:

The configurations made in the *Configure Message Blocks* table of the previous sample are considered complete and will not be described separately again.

Creation of a new Message Window Template

Step	Procedure: Creation of a new Message Window Template
1	<p>Open the <i>Alarm Logging</i> editor.</p> <p>If a message comes in, the message window is to display the current value of the tag monitored. For this, a new process value block must be created.</p> <p>Open the <i>Configure Message Blocks</i> dialog via a  on the <i>Message Blocks</i> entry. In the dialog, select the <i>Process Value Blocks</i> list entry and open the <i>Add Process Value Blocks...</i> dialog by clicking on the Add button. In this dialog, a new process value block is added. The dialog is closed by clicking on <i>OK</i>.</p> <p>Via a  on the <i>Process Value Blocks</i> list entry, the new block will be displayed. If this block is selected, its properties dialog can be accessed via the properties button. In this sample, <i>Value</i> was entered as the <i>Name</i> of the block and 5 characters as its <i>Length</i>.</p> <p>By clicking on <i>OK</i>, the settings made in the <i>Message Blocks</i> and <i>Configure Message Blocks</i> dialogs are applied.</p> 



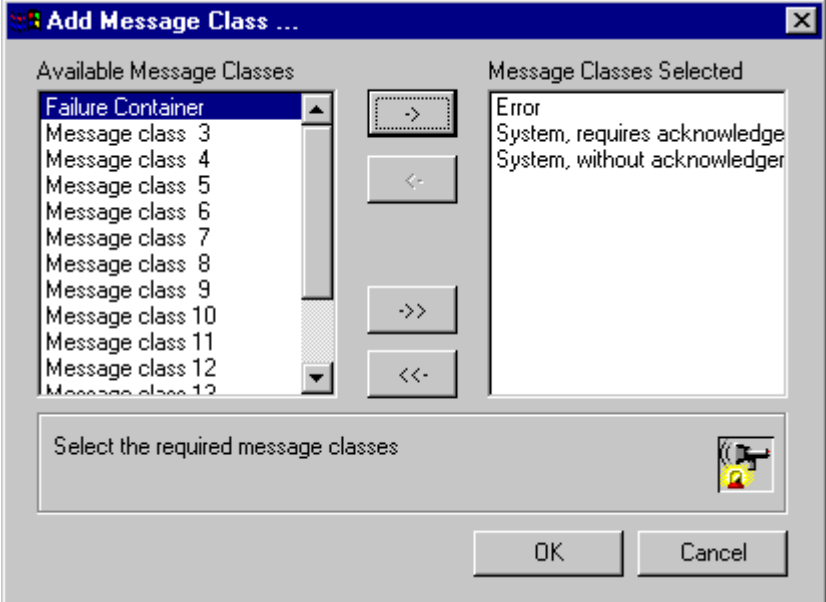


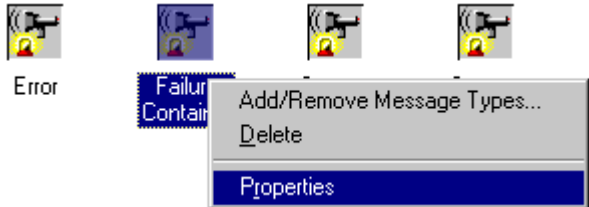
General Information

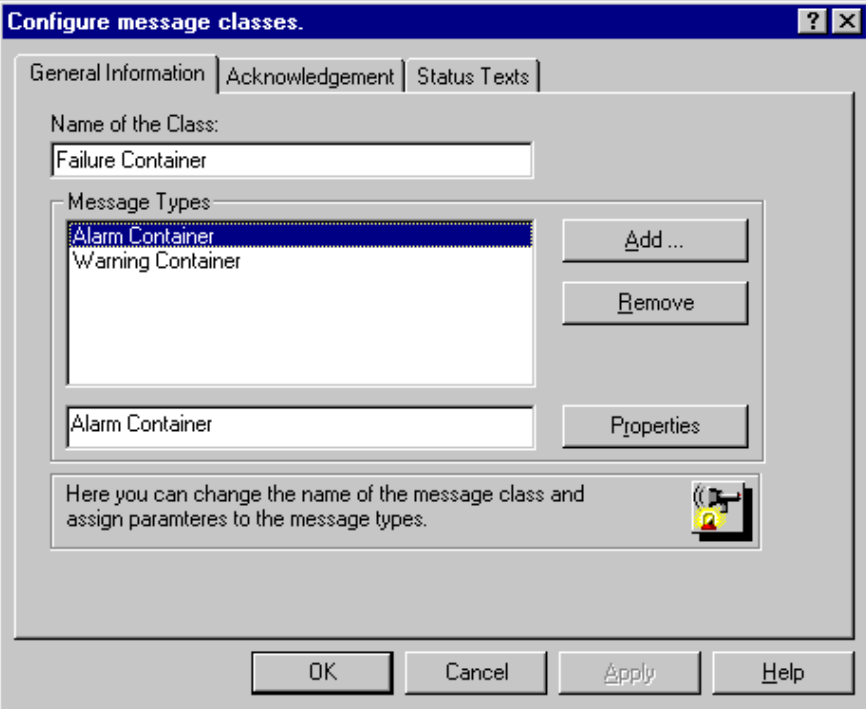
With the help of message classes,

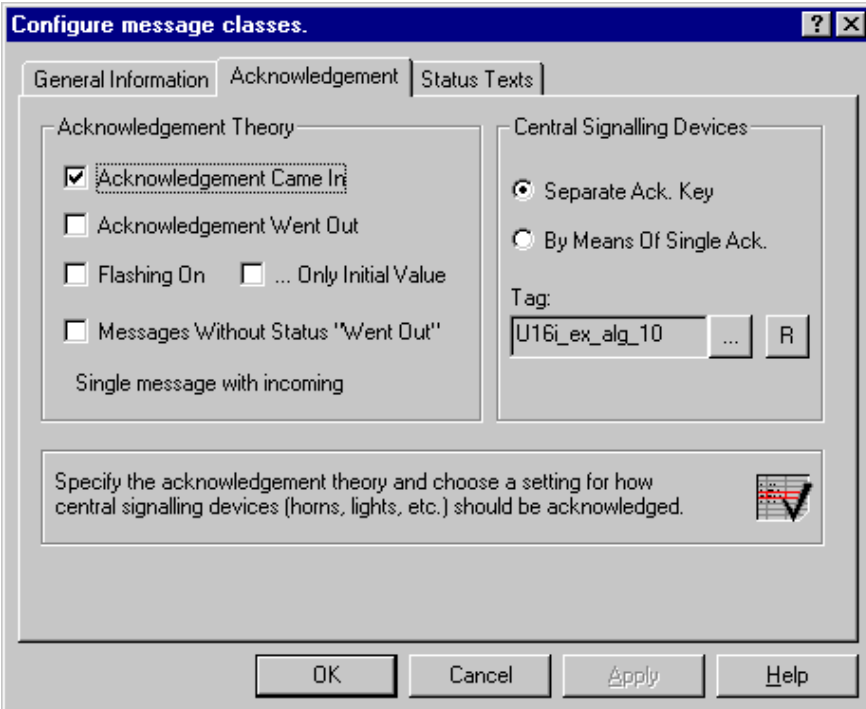
- the acknowledgment type
- the corresponding status text
- the output of acoustical/optical signals

is specified for all message types belonging to a message class.

Creation of a new Message Class




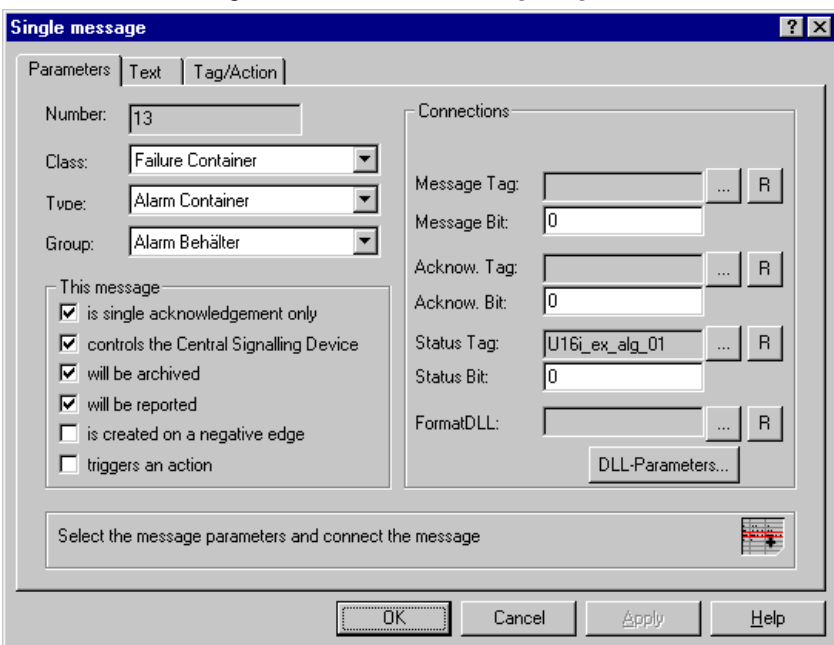
Step	Procedure: Creation of a new Message Class
1	<p>Via a  on the <i>Message Class</i> entry, open the <i>Add Message Class...</i> dialog.</p> 
2	<p>Via the -> button, a new message class is added. Close the dialog box by clicking on <i>OK</i>.</p> 
3	<p>Via a  on the <i>Message Class</i> entry, all created message classes, even the newly added one, are displayed. Via a  on its icon, the <i>Configure Message Class</i> dialog can be opened.</p> 

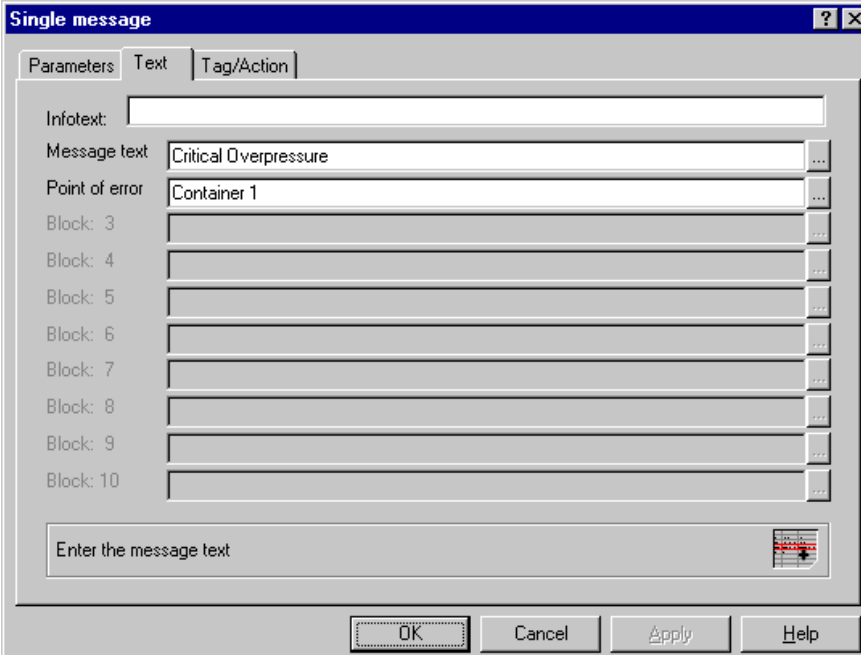
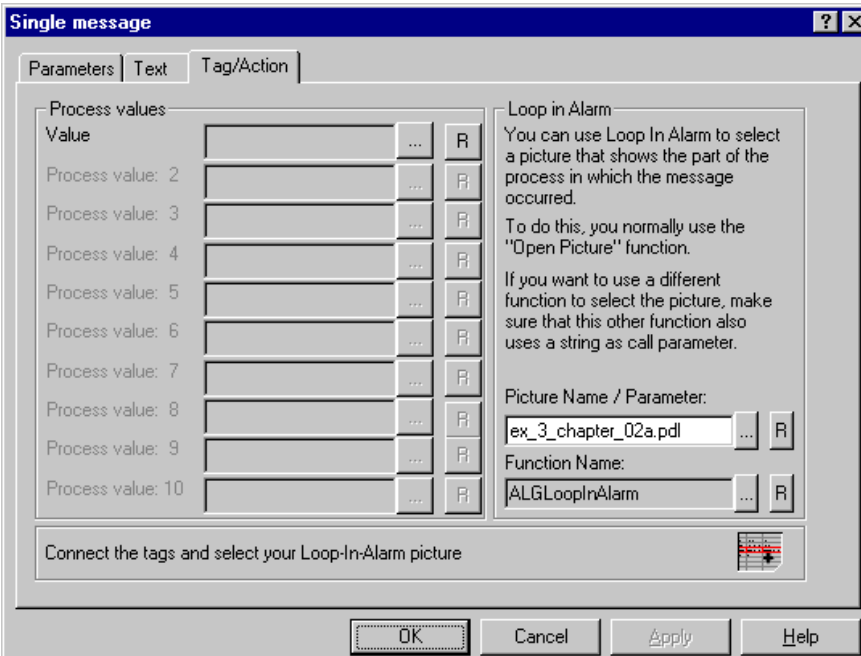
Step	Procedure: Creation of a new Message Class
4	<p>In the <i>General Information</i> tab, as the <i>Name of the Class</i>, <i>Container Error</i> is entered.</p> <p>Via the add button, the <i>Add Message Type...</i> dialog is accessed. In this dialog, two message types from the left window are moved to the right window via the <i>-></i> button. Close the dialog box by clicking on <i>OK</i>.</p> <p>If one of the new message types is selected in the <i>Message Types</i> field, its properties dialog can be opened via the properties button.</p> <p>As the name of the first message type, <i>Container Alarm</i> is entered. The color scheme of the individual message stati looks as follows:</p> <p>Came in: Text = black, Background = red Went out: Text = black, Background = green Acknowledged: Text = black, Background = orange</p> <p>As the name of the second message type, <i>Container Warning</i> is entered. The color scheme of the individual message stati looks as follows:</p> <p>Came in: Text = yellow, Background = blue Went out: Text = blue, Background = RGB(207,163,146) Acknowledged: Text = white, Background = blue</p> 

Step	Procedure: Creation of a new Message Class
5	<p>In the <i>Acknowledgment</i> tab, the <i>Acknowledgment Came In</i> check-box is selected from the <i>Acknowledgment Philosophy</i> field.</p> <p>In the <i>Acknowledgment of Central Indicator</i> field, the <i>Separate Acknowledgment Key</i> radio-button is selected. As the <i>Tag</i>, the <i>U16i_ex_alg_10</i> tag is set. This tag controls a central indicator. In order to acknowledge this indicator, a separate button must be configured on the toolbar. If a standard toolbar is configured, this is the <i>Horn Acknowledgment</i> button.</p>  <p>The screenshot shows a dialog box titled "Configure message classes." with three tabs: "General Information", "Acknowledgement", and "Status Texts". The "Acknowledgement" tab is active. It contains two main sections: "Acknowledgement Theory" and "Central Signalling Devices". In "Acknowledgement Theory", the "Acknowledgement Came In" checkbox is checked, while "Acknowledgement Went Out", "Flashing On", and "Messages Without Status 'Went Out'" are unchecked. There are also options for "... Only Initial Value" and "Single message with incoming". In "Central Signalling Devices", the "Separate Ack. Key" radio button is selected, and "By Means Of Single Ack." is unselected. Below this, the "Tag" field contains the text "U16i_ex_alg_10" and has a selection button with "R". At the bottom of the dialog are "OK", "Cancel", "Apply", and "Help" buttons.</p>
6	<p>In the <i>Status Texts</i> tab, no additional settings are made.</p> <p>Close the dialog box by clicking on <i>OK</i>.</p>

4.2.3 Limit Value Monitoring (Continuation)

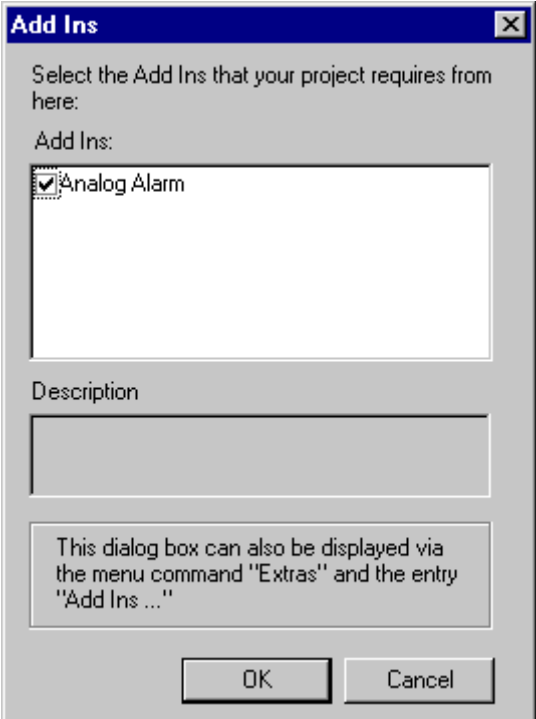


Creation of Single Messages

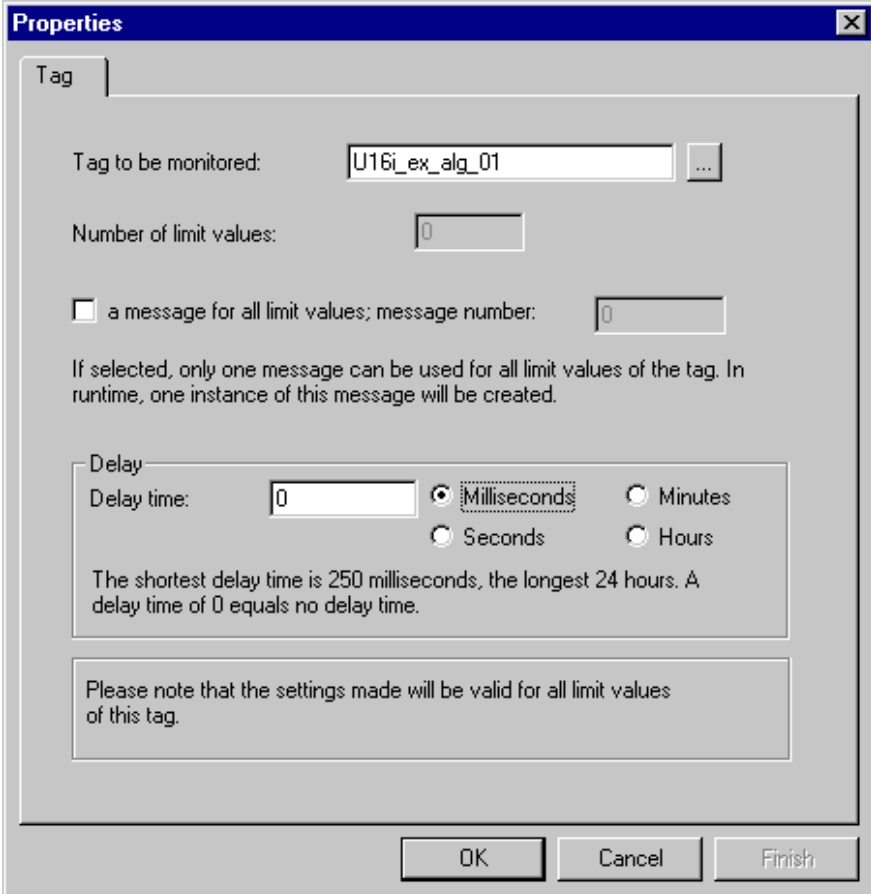


Step	Procedure: Creation of Single Messages																																																																				
1	<p>Via a  in the table window, 12 new lines are added.</p> <table border="1"> <thead> <tr> <th>Number</th> <th>Class</th> <th>Type</th> <th>Message Tag</th> </tr> </thead> <tbody> <tr><td>1</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_00</td></tr> <tr><td>2</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_00</td></tr> <tr><td>3</td><td>Error</td><td></td><td>_ex_alg_00</td></tr> <tr><td>4</td><td>Error</td><td></td><td>_ex_alg_03</td></tr> <tr><td>5</td><td>Error</td><td></td><td>_ex_alg_03</td></tr> <tr><td>6</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_03</td></tr> <tr><td>7</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_03</td></tr> </tbody> </table> <p>The first of the newly added lines is selected via the . Via a  on this line, the <i>Single Messages</i> dialog can be opened.</p> <table border="1"> <tbody> <tr><td>4</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_</td></tr> <tr><td>5</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>6</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>7</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>8</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>9</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>10</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>11</td><td></td><td></td><td>U16i_ex_alg_</td></tr> <tr><td>12</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_</td></tr> </tbody> </table>	Number	Class	Type	Message Tag	1	Error	Failure	U16i_ex_alg_00	2	Error	Failure	U16i_ex_alg_00	3	Error		_ex_alg_00	4	Error		_ex_alg_03	5	Error		_ex_alg_03	6	Error	Failure	U16i_ex_alg_03	7	Error	Failure	U16i_ex_alg_03	4	Error	Failure	U16i_ex_alg_	5			U16i_ex_alg_	6			U16i_ex_alg_	7			U16i_ex_alg_	8			U16i_ex_alg_	9			U16i_ex_alg_	10			U16i_ex_alg_	11			U16i_ex_alg_	12	Error	Failure	U16i_ex_alg_
Number	Class	Type	Message Tag																																																																		
1	Error	Failure	U16i_ex_alg_00																																																																		
2	Error	Failure	U16i_ex_alg_00																																																																		
3	Error		_ex_alg_00																																																																		
4	Error		_ex_alg_03																																																																		
5	Error		_ex_alg_03																																																																		
6	Error	Failure	U16i_ex_alg_03																																																																		
7	Error	Failure	U16i_ex_alg_03																																																																		
4	Error	Failure	U16i_ex_alg_																																																																		
5			U16i_ex_alg_																																																																		
6			U16i_ex_alg_																																																																		
7			U16i_ex_alg_																																																																		
8			U16i_ex_alg_																																																																		
9			U16i_ex_alg_																																																																		
10			U16i_ex_alg_																																																																		
11			U16i_ex_alg_																																																																		
12	Error	Failure	U16i_ex_alg_																																																																		
2	<p>In the <i>Parameters</i> tab, select the message class <i>Container Error</i> and the message type <i>Container Alarm</i>. In the <i>This Message</i> field, the <i>is single acknowledgment</i>, <i>controls the horn</i>, <i>will be archived</i> and <i>will be reported</i> check-boxes are selected. In the <i>Connections</i> field, select as the <i>Status Tag</i> the tag <i>U16i_ex_alg_01</i>. As the <i>Status Bit</i>, 0 is entered. No <i>Event Tag</i> is set, since the message is generated by the limit value monitoring. Likewise, no <i>Acknowledge Tag</i> is set.</p> 																																																																				

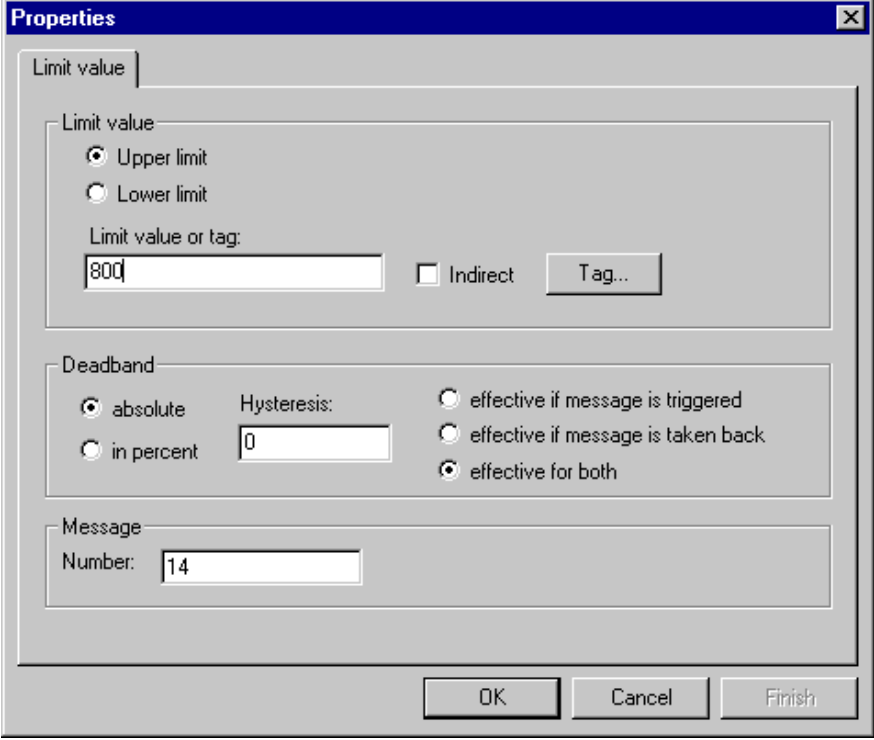

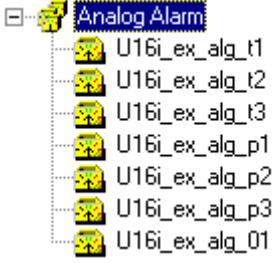
Step	Procedure: Creation of Single Messages
3	<p>In the <i>Text</i> tab, the <i>Message Text Critical Overpressure</i> and the <i>Point of Error Container 1</i> are entered. As the <i>Infotext</i>, <i>The pressure in container 1 has exceeded the critical value</i> is entered.</p> 
4	<p>In the <i>Tag/Action</i> tab, a tag could be set for the <i>Value</i> process value block. However, if the message is generated via the limit value monitoring, the first process value block of the message line is automatically supplied with the limit value that caused the message to be triggered.</p> <p>The settings made are applied by clicking on <i>OK</i>.</p> 

Step	Procedure: Creation of Single Messages																																																																																																																																																																																													
5	<p>The just created message monitors the pressure in the first of the three containers. For the first container, three more message lines are created.</p> <p>The settings are made as described in step 2, however, for the additional message of the message type <i>Container Error</i>, the <i>Message Text Critical Temperature</i> and a correspondingly changed <i>Infotext</i> are entered. In addition, two messages of the message type <i>Container Warning</i> are created, which have the <i>Message Texts Pressure Warning</i> and <i>Temperature Warning</i>. For these warnings, all check-boxes in the <i>This Message</i> field in the <i>Single Message</i> dialog of the <i>Parameters</i> tab are deselected. For all messages pertaining to container 1, the same status tag, but with an adapted status bit, is used.</p>																																																																																																																																																																																													
6	<p>For the other two containers, four messages each are also created.</p> <p>Here, the <i>Status Tags</i> and the texts for the <i>Point of Error</i> must be adapted to the respective containers.</p> <table border="1"> <thead> <tr> <th>Number</th> <th>Class</th> <th>Type</th> <th>MessageTag</th> <th>MessageBit</th> <th>Status tag</th> <th>Status bit</th> <th>Message text</th> <th>Point of err</th> </tr> </thead> <tbody> <tr><td>5</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_03</td><td>1</td><td>U16i_ex_alg_05</td><td>1</td><td>Feedback Error</td><td>Motor 2</td></tr> <tr><td>6</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_03</td><td>2</td><td>U16i_ex_alg_05</td><td>2</td><td>Bimetal Error</td><td>Motor 2</td></tr> <tr><td>7</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_06</td><td>0</td><td>U16i_ex_alg_08</td><td>0</td><td>Lock Error</td><td>Motor 3</td></tr> <tr><td>8</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_06</td><td>1</td><td>U16i_ex_alg_08</td><td>1</td><td>Feedback Error</td><td>Motor 3</td></tr> <tr><td>9</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_06</td><td>2</td><td>U16i_ex_alg_08</td><td>2</td><td>Bimetal Error</td><td>Motor 3</td></tr> <tr><td>10</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_09</td><td>0</td><td>U16i_ex_alg_11</td><td>0</td><td>Lock Error</td><td>Motor 4</td></tr> <tr><td>11</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_09</td><td>1</td><td>U16i_ex_alg_11</td><td>1</td><td>Feedback Error</td><td>Motor 4</td></tr> <tr><td>12</td><td>Error</td><td>Failure</td><td>U16i_ex_alg_09</td><td>2</td><td>U16i_ex_alg_11</td><td>2</td><td>Bimetal Error</td><td>Motor 4</td></tr> <tr><td>13</td><td>Failure Container</td><td>Alarm Contain</td><td></td><td>0</td><td>U16i_ex_alg_01</td><td>0</td><td>Critical Overpressure</td><td>Container 1</td></tr> <tr><td>14</td><td>Failure Container</td><td>Alarm Contain</td><td></td><td>0</td><td>U16i_ex_alg_01</td><td>1</td><td>Critical Temperature</td><td>Container 1</td></tr> <tr><td>15</td><td>Failure Container</td><td>Alarm Contain</td><td></td><td>0</td><td>U16i_ex_alg_04</td><td>0</td><td>Critical Overpressure</td><td>Container 2</td></tr> <tr><td>16</td><td>Failure Container</td><td>Alarm Contain</td><td></td><td>0</td><td>U16i_ex_alg_04</td><td>1</td><td>Critical Temperature</td><td>Container 2</td></tr> <tr><td>17</td><td>Failure Container</td><td>Alarm Contain</td><td></td><td>0</td><td>U16i_ex_alg_07</td><td>0</td><td>Critical Overpressure</td><td>Container 3</td></tr> <tr><td>18</td><td>Failure Container</td><td>Alarm Contain</td><td></td><td>0</td><td>U16i_ex_alg_07</td><td>1</td><td>Critical Temperature</td><td>Container 3</td></tr> <tr><td>19</td><td>Failure Container</td><td>Warning Cont</td><td></td><td>0</td><td>U16i_ex_alg_01</td><td>2</td><td>Warning pressure</td><td>Container 1</td></tr> <tr><td>20</td><td>Failure Container</td><td>Warning Cont</td><td></td><td>0</td><td>U16i_ex_alg_01</td><td>3</td><td>Warning temperature</td><td>Container 1</td></tr> <tr><td>21</td><td>Failure Container</td><td>Warning Cont</td><td></td><td>0</td><td>U16i_ex_alg_04</td><td>2</td><td>Warning pressure</td><td>Container 2</td></tr> <tr><td>22</td><td>Failure Container</td><td>Warning Cont</td><td></td><td>0</td><td>U16i_ex_alg_04</td><td>3</td><td>Warning temperature</td><td>Container 2</td></tr> <tr><td>23</td><td>Failure Container</td><td>Warning Cont</td><td></td><td>0</td><td>U16i_ex_alg_07</td><td>2</td><td>Warning pressure</td><td>Container 3</td></tr> <tr><td>24</td><td>Failure Container</td><td>Warning Cont</td><td></td><td>0</td><td>U16i_ex_alg_07</td><td>3</td><td>Warning temperature</td><td>Container 3</td></tr> </tbody> </table>	Number	Class	Type	MessageTag	MessageBit	Status tag	Status bit	Message text	Point of err	5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error	Motor 2	6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error	Motor 2	7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error	Motor 3	8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error	Motor 3	9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error	Motor 3	10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error	Motor 4	11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error	Motor 4	12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error	Motor 4	13	Failure Container	Alarm Contain		0	U16i_ex_alg_01	0	Critical Overpressure	Container 1	14	Failure Container	Alarm Contain		0	U16i_ex_alg_01	1	Critical Temperature	Container 1	15	Failure Container	Alarm Contain		0	U16i_ex_alg_04	0	Critical Overpressure	Container 2	16	Failure Container	Alarm Contain		0	U16i_ex_alg_04	1	Critical Temperature	Container 2	17	Failure Container	Alarm Contain		0	U16i_ex_alg_07	0	Critical Overpressure	Container 3	18	Failure Container	Alarm Contain		0	U16i_ex_alg_07	1	Critical Temperature	Container 3	19	Failure Container	Warning Cont		0	U16i_ex_alg_01	2	Warning pressure	Container 1	20	Failure Container	Warning Cont		0	U16i_ex_alg_01	3	Warning temperature	Container 1	21	Failure Container	Warning Cont		0	U16i_ex_alg_04	2	Warning pressure	Container 2	22	Failure Container	Warning Cont		0	U16i_ex_alg_04	3	Warning temperature	Container 2	23	Failure Container	Warning Cont		0	U16i_ex_alg_07	2	Warning pressure	Container 3	24	Failure Container	Warning Cont		0	U16i_ex_alg_07	3	Warning temperature	Container 3
Number	Class	Type	MessageTag	MessageBit	Status tag	Status bit	Message text	Point of err																																																																																																																																																																																						
5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error	Motor 2																																																																																																																																																																																						
6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error	Motor 2																																																																																																																																																																																						
7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error	Motor 3																																																																																																																																																																																						
8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error	Motor 3																																																																																																																																																																																						
9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error	Motor 3																																																																																																																																																																																						
10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error	Motor 4																																																																																																																																																																																						
11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error	Motor 4																																																																																																																																																																																						
12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error	Motor 4																																																																																																																																																																																						
13	Failure Container	Alarm Contain		0	U16i_ex_alg_01	0	Critical Overpressure	Container 1																																																																																																																																																																																						
14	Failure Container	Alarm Contain		0	U16i_ex_alg_01	1	Critical Temperature	Container 1																																																																																																																																																																																						
15	Failure Container	Alarm Contain		0	U16i_ex_alg_04	0	Critical Overpressure	Container 2																																																																																																																																																																																						
16	Failure Container	Alarm Contain		0	U16i_ex_alg_04	1	Critical Temperature	Container 2																																																																																																																																																																																						
17	Failure Container	Alarm Contain		0	U16i_ex_alg_07	0	Critical Overpressure	Container 3																																																																																																																																																																																						
18	Failure Container	Alarm Contain		0	U16i_ex_alg_07	1	Critical Temperature	Container 3																																																																																																																																																																																						
19	Failure Container	Warning Cont		0	U16i_ex_alg_01	2	Warning pressure	Container 1																																																																																																																																																																																						
20	Failure Container	Warning Cont		0	U16i_ex_alg_01	3	Warning temperature	Container 1																																																																																																																																																																																						
21	Failure Container	Warning Cont		0	U16i_ex_alg_04	2	Warning pressure	Container 2																																																																																																																																																																																						
22	Failure Container	Warning Cont		0	U16i_ex_alg_04	3	Warning temperature	Container 2																																																																																																																																																																																						
23	Failure Container	Warning Cont		0	U16i_ex_alg_07	2	Warning pressure	Container 3																																																																																																																																																																																						
24	Failure Container	Warning Cont		0	U16i_ex_alg_07	3	Warning temperature	Container 3																																																																																																																																																																																						

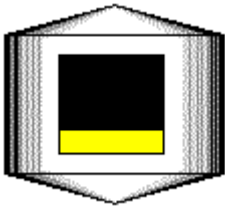

Configuration of the Limit Value Monitoring


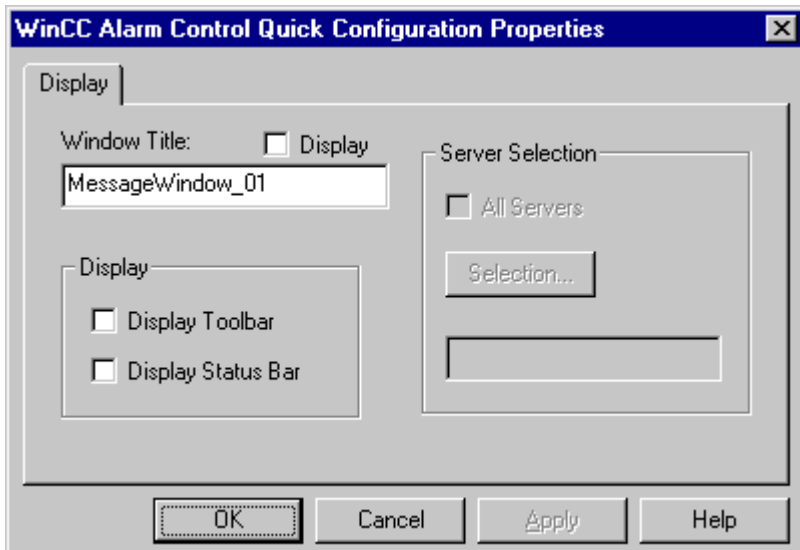
Step	Procedure: Configuration of the Limit Value Monitoring
1	<p>If the <i>Limit Value Monitoring</i> (Analog Alarm) entry is not present in the navigation window, it must be loaded. This is done via the <i>Options</i> → <i>Add Ins</i> menus in <i>Alarm Logging</i>. In the displayed dialog, the entry for the limit value monitoring must be check-marked.</p> 
2	<p>Via a  on the <i>Limit Value Monitoring</i> entry and then selecting <i>New</i> from the pop-up menu, the <i>Properties</i> dialog of the tag is accessed. In this dialog, a new tag for the limit value monitoring can be set.</p> 


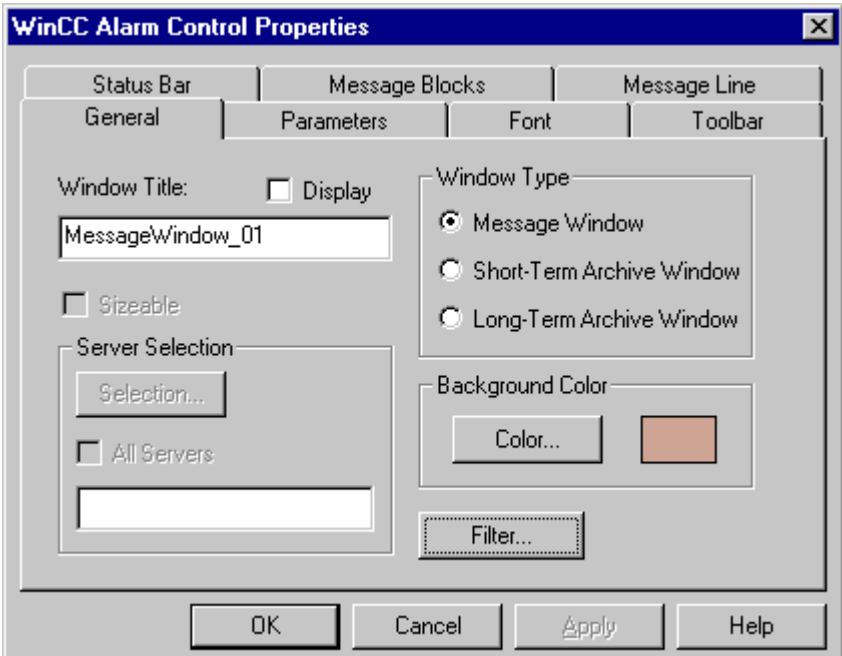
Step	Procedure: Configuration of the Limit Value Monitoring
3	<p>In this dialog, the <i>U16i_ex_alg_t1</i> tag containing the temperature value of the first container is set as the <i>Tag to be monitored</i>. The check-box <i>a message for all limit values</i> is not selected. As the <i>Delay Time</i>, 0 is kept.</p> <p>Exit the dialog box by clicking on <i>OK</i>.</p> 
4	<p>In the right window, the icon of the tag to be monitored is displayed. Via a  on this icon and then selecting <i>New</i> from the pop-up menu, the <i>Properties</i> dialog of the limit value is opened. In this dialog, a new limit value can be assigned to the tag.</p> 


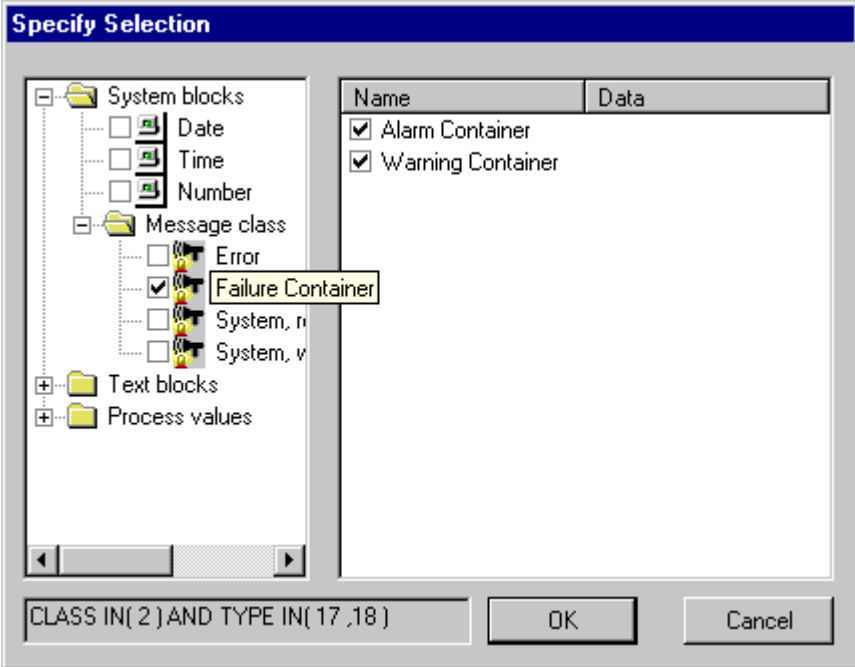
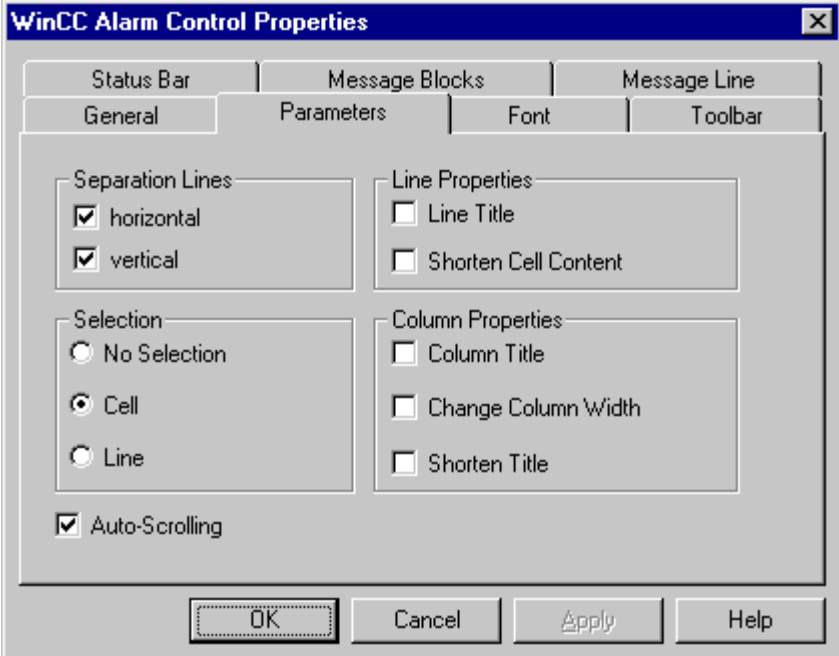
Step	Procedure: Configuration of the Limit Value Monitoring
5	<p>In the <i>Limit Value</i> field, the <i>Upper Limit</i> is set. In the <i>Limit Value or Tag</i> field, a limit value of 800 is entered. As the <i>Hysteresis</i>, 0 is kept. As the <i>Message</i>, the <i>Number 14</i> is entered. This is the alarm message if the temperature in the first container is exceeded.</p> <p>Exit the dialog box by clicking on <i>OK</i>.</p> <p>For the same tag, a second limit value is specified. In the <i>Limit Value</i> field, an <i>Upper Limit</i> is set again. However, in the <i>Limit Value or Tag</i> field, a limit value of 500 is entered. As the <i>Message</i>, the <i>Number 20</i> is entered. This is the warning message if the temperature in the first container is exceeded.</p> 
6	<p>The remaining five tags to be monitored are created as described in steps 2 and 3, each with two configured limit values.</p> <p>Via a  on the <i>Limit Value Monitoring</i> entry, all created tags will be displayed.</p> 


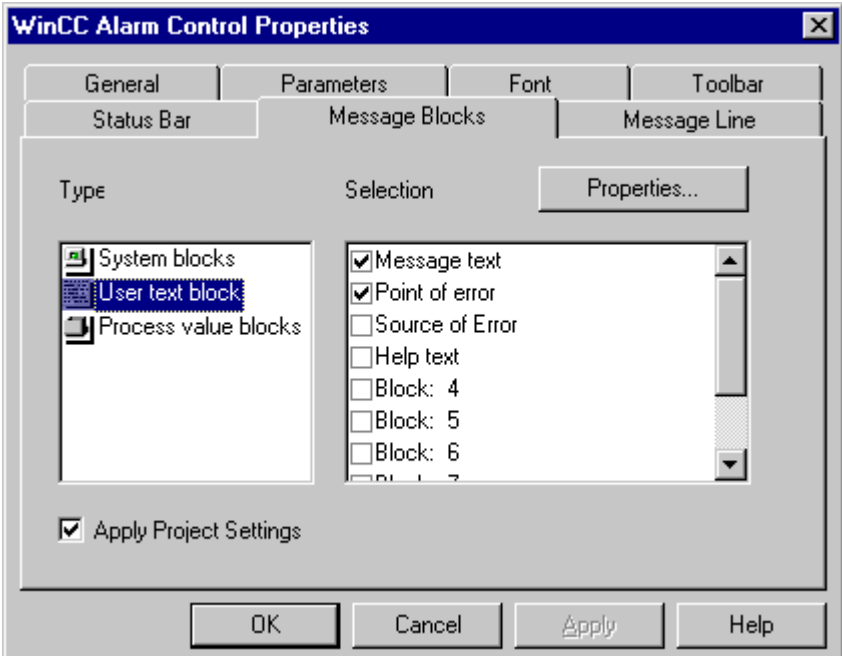
Implementation in the Graphics Designer

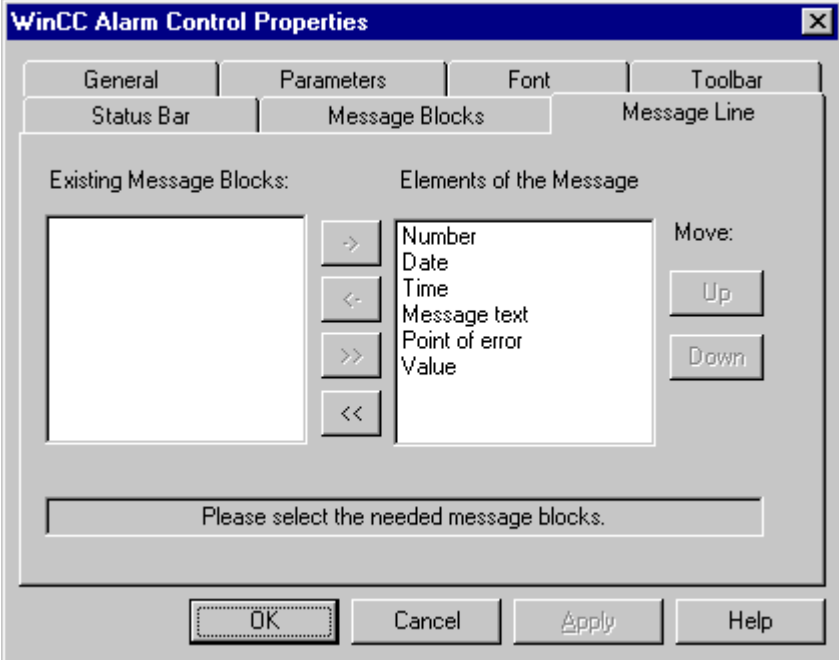


Step	Procedure: Implementation in the Graphics Designer
1	<p>The simulation of the process values to be monitored is implemented via a <i>Windows Object</i> → <i>Slider Object</i> each. In the sample, these are the <i>Slider Object1</i> to <i>Slider Object6</i>.</p> <p>For <i>Slider Object1</i>, create a <i>Direct Connection</i> at <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Process Driver Connection</i> that switches the current process value of the <i>Slider</i> to the <i>U16i_ex_alg_t1</i> tag. This <i>Slider</i> simulates the temperature value in the first container. In the same manner, configure a <i>Slider</i> for the remaining tags.</p> <p>To match the slider position with the current tag value upon opening the picture, a <i>C-Action</i> is created at <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> that performs this task.</p>
2	<p>in addition, a <i>Smart Object</i> → <i>I/O Field</i> is assigned to each <i>Slider</i> to display the current tag value. In the sample, these are the <i>I/O Field1</i> to <i>I/O Field6</i> objects.</p> <p>For <i>I/O Field1</i>, create <i>Tag Connection</i> at <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i> to the <i>U16i_ex_alg_t1</i> tag and trigger it upon change. This is the <i>I/O Field</i> assigned to the first <i>Slider</i>. In the same manner, assign an <i>I/O Field</i> to each remaining <i>Slider</i>.</p>
3	<p>The display of the individual containers is realized via the <i>Smart Object Tank4</i> from the standard library. In this sample, these are the <i>Tank41</i>, <i>Tank42</i> and <i>Tank43</i> objects.</p> <p>These objects are only used for display purposes and receive no dynamics.</p> 
4	<p>A <i>Smart Object</i> → <i>Status Display</i> is assigned to each container that displays a <i>warning light</i>. In this sample, these are the <i>Status Display1</i> to <i>Status Display3</i> objects.</p> <p>In this sample, the bitmap <i>Blinker blinkt nicht.gif</i> is set as the <i>Basic Picture</i> and the bitmap <i>Bliker blinkt.gif</i> as the <i>Flash Picture</i> of <i>Status Display1</i> at <i>Properties</i> → <i>State</i> for the <i>Current Status 0</i>. The <i>Property</i> → <i>State</i> → <i>Flashing Flash Picture Active</i> is set to <i>no</i>. For the same property, a <i>C-Action</i> is created that activates the flashing if an alarm message for the corresponding container is pending. The other two <i>Status Displays</i> are configured in the same manner.</p> 


Step	Procedure: Implementation in the Graphics Designer
5	<p>An additional <i>Smart Object</i> → <i>Status Display</i> is configured that displays a horn. In the sample, this is the <i>Status Display4</i> object. The bitmap <i>Hupe hupt nicht.gif</i> is set as the <i>Basic Picture</i> and the bitmap <i>Hupe hupt.gif</i> as the <i>Flash Picture</i> of this object at <i>Properties</i> → <i>State</i> for the <i>Current Status 0</i>. The <i>Property</i> → <i>State</i> → <i>Flashing Flash Picture Active</i> is set to <i>no</i>. At the same property, create a <i>C-Action</i> that activates the flashing if an alarm message pertaining to one of the three containers comes in, i.e. if the tag set in <i>Alarm Logging</i> for the <i>Container Error</i> message class, controlling the central indicator, takes on the status <i>1</i>. In the sample, this is the <i>U16i_ex_alg_10</i> tag.</p> <p>At <i>Properties</i> → <i>Geometry</i> → <i>Width</i>, a <i>C-Action</i> is created that emits audible signals if the object is flashing.</p> 
6	<p>To display the messages configured in <i>Alarm Logging</i>, a <i>WinCC Alarm Control</i> is used. It is selected from the <i>Object Palette's Control</i> selection menu and then placed in the picture.</p>
7	<p>After placing the <i>Control</i> in the picture, its <i>configuration dialog</i> will be displayed automatically.</p> <p>As the <i>Window Title</i>, <i>MessageWindow_01</i> is entered. The <i>Display</i> check-box remains deselected. In the <i>C-Actions</i> created later, this window title is used to reference the corresponding <i>Control</i>.</p> <p>The <i>Toolbar</i> and <i>Status Bar</i> check-boxes are deselected.</p> <p>The <i>configuration dialog</i> can be exited by clicking on <i>OK</i>.</p> 

Step	Procedure: Implementation in the Graphics Designer
8	<p>Open the Control's <i>Properties</i> dialog. This dialog is displayed via a  on the Control. In the <i>General Information</i> tab, the selection button is used to match the background color to the existing project color scheme.</p> <p>The <i>Selection</i> button is used to select the single messages, previously created in Alarm Logging, to be displayed by the Control.</p> 

Step	Procedure: Implementation in the Graphics Designer						
9	<p>Via a  on the <i>System Block Message Class</i> → <i>Container Error</i>, 2 check-boxes are displayed in the right window. The check-boxes <i>Container Alarm</i> and <i>Container Warning</i> are selected. This means that in runtime only messages of the message class container error will be displayed in the message window.</p>  <p>Specify Selection</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/> Alarm Container</td> <td></td> </tr> <tr> <td><input checked="" type="checkbox"/> Warning Container</td> <td></td> </tr> </tbody> </table> <p>CLASS IN{ 2 } AND TYPE IN{ 17 ,18 }</p> <p>OK Cancel</p>	Name	Data	<input checked="" type="checkbox"/> Alarm Container		<input checked="" type="checkbox"/> Warning Container	
Name	Data						
<input checked="" type="checkbox"/> Alarm Container							
<input checked="" type="checkbox"/> Warning Container							
10	<p>In the <i>Parameters</i> tab, the check-boxes <i>Line Title</i>, <i>Column Title</i> and <i>Change Column Width</i> are deselected. In the <i>Selection</i> field, the <i>Cell</i> radio-button is selected.</p>  <p>WinCC Alarm Control Properties</p> <p>Status Bar Message Blocks Message Line</p> <p>General Parameters Font Toolbar</p> <p>Separation Lines</p> <p><input checked="" type="checkbox"/> horizontal</p> <p><input checked="" type="checkbox"/> vertical</p> <p>Selection</p> <p><input type="radio"/> No Selection</p> <p><input checked="" type="radio"/> Cell</p> <p><input type="radio"/> Line</p> <p>Line Properties</p> <p><input type="checkbox"/> Line Title</p> <p><input type="checkbox"/> Shorten Cell Content</p> <p>Column Properties</p> <p><input type="checkbox"/> Column Title</p> <p><input type="checkbox"/> Change Column Width</p> <p><input type="checkbox"/> Shorten Title</p> <p><input checked="" type="checkbox"/> Auto-Scrolling</p> <p>OK Cancel Apply Help</p>						

Step	Procedure: Implementation in the Graphics Designer
11	<p>In the <i>Message Blocks</i> tab, the columns are selected that will later be displayed in the message line. In this sample, <i>system blocks</i> are selected in the <i>Type</i> field using the . In the right window, <i>Date</i>, <i>Time</i> and <i>Number</i> are selected. For the <i>User Text Blocks</i> entry, <i>Message Text</i> and <i>Point of Error</i> are selected. For the <i>Process Value Blocks</i> entry, <i>Value</i> is selected.</p> 

Step	Procedure: Implementation in the Graphics Designer
12	<p>In the <i>Message Line</i> tab, the previously selected <i>Message Blocks</i> are assigned to the message line. The <i>Available Message Blocks</i> fields lists the available columns. By pressing on the -> button, each message block can individually be added to the message line. Via the >> button, all message blocks listed in the window can be assigned to the message line at one time. The properties dialog is exited by clicking on <i>OK</i>.</p> 
13	<p>For the toolbar, several <i>Windows Objects</i> → <i>Buttons</i> are configured that simulate the pressing of the individual buttons via special standard functions.</p>
14	<p>A <i>Button</i> for the single acknowledgment of a message is configured. This button also acknowledges the horn, if it has been triggered. The associated standard functions are:</p> <pre>AXC_OnBtnSinglAckn(lpszPictureName,lpszObjectName) AXC_OnBtnHornAckn(lpszPictureName,lpszObjectName)</pre> 
15	<p>Additional buttons are configured. One <i>Button</i> for the group acknowledgment and one <i>Button</i> for calling the Infotext dialog. The associated standard functions are:</p> <pre>AXC_OnBtnVisibleAckn(lpszPictureName,lpszObjectName) AXC_OnBtnInfo(lpszPictureName,lpszObjectName)</pre> 

Step	Procedure: Implementation in the Graphics Designer
16	<p>As the replacement for the button that turns the auto-scroll function on and off, a <i>Smart Object</i> → <i>Status Display</i> is used. In this sample, this is the <i>Status Display6</i> object.</p> <p>At <i>Properties</i> → <i>State</i> → <i>Current Status</i>, a <i>Tag Connection</i> to the <i>BINi_ex_alg_00</i> tag is created. This tag contains the information whether auto-scrolling is turned on or off. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, a <i>C-Action</i> is created that inverts the status of the <i>BINi_ex_alg_00</i> tag and calls the standard function <i>AXC_OnBtmScroll</i>(lpszPictureName,lpszObjectName). At the opening of the picture, the <i>BINi_ex_alg_00</i> tag is set to 0, since auto-scroll is turned if a message window is reselected.</p> 

C-Action at Status Display1

```
#include "apdefap.h"
BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty,
{
WORD state;

state=GetTagWord("U16i_ex_alg_01");

if ((state&1)|| (state&2)) return TRUE;
else return FALSE;
}
```

- Reading of the status tag of the first container. If an alarm message is pending, *TRUE* will be returned to the property and the warning lamp flashes.
- This *C-Action* is triggered upon the change of the status tag of the first container.

C-Action at Status Display4

```
#include "apdefap.h"
BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty,
{
if (GetTagWord("U16i_ex_alg_10")&1) return TRUE;
else return FALSE;
}
```

-
-
- If the central indicator is triggered, *TRUE* will be returned to the property and the horn will be displayed optically.
- This *C-Action* is triggered upon the change of the tag controlling the central indicator.

C-Action for Generating Audible Signals

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
#pragma code ("winmm.dll")
BOOL PlaySound(LPCSTR pszSound, HMODULE hMod, DWORD fdwSound);
#define SND_FILENAME 0x00020000L
#define SND_ASYNC 0x0001
#pragma code ()

char szProjectName[MAX_PATH];
CMN_ERROR Error;
char szSoundFilePath[MAX_PATH] = "";
char szSoundFile[MAX_PATH] = "Hupe.wav";

if (GetFlashFlashPicture(lpszPictureName, lpszObjectName)) {
    if (DMGetRuntimeProject( szProjectName, MAX_PATH, &Error)) {
        strcat(szSoundFilePath, szProjectName,
            strlen(szProjectName)-strlen(strrchr(szProjectName, '\\))+1));
        strcat(szSoundFilePath, szSoundFile);
        //MessageBeep((WORD)-1);
        PlaySound(szSoundFilePath, NULL, SND_FILENAME| SND_ASYNC);
    }
}

return 56;
}
```

- Loading of the DLL *winmm.dll*. This DLL contains the function for playing sound files.
- If the *Status Display4* object flashes, the *Hupe.wav* file, which is located in the project folder, is to be played. For this, the project folder must be determined via the *DMGetRuntimeProject* function and the file path be assembled.
- Call of the *PlaySound* function.
- This *C-Action* is executed in one second cycles.

Note for the General Application

The following adaptations must be made before the general application:

- The message class created must be adapted to meet your own requirements.
- The display type of the message window must adapted to meet your requirements.

4.2.4 Message Window (ex_3_chapter_02b.pdl)

Task Definition

Via a message window, multiple processes are to be monitored. If a message comes in, a button on the toolbar should make it possible to jump to the window, where the error occurred.

The message window is created using the standard tools of *Alarm Logging*, the standard toolbar and the standard status bar are to be used.

Implementation Concept


This sample uses the messages and pictures created in the previous samples. A project function is needed that performs the picture change if the *Loop In Alarm* button on the toolbar is pressed.



The message window is created in the *Graphics Designer* using a WinCC Alarm Control. No additional objects are needed.

Note:


The configurations made in the previous two samples are considered as complete. They will not be explained separately again, however, this sample is based on them.

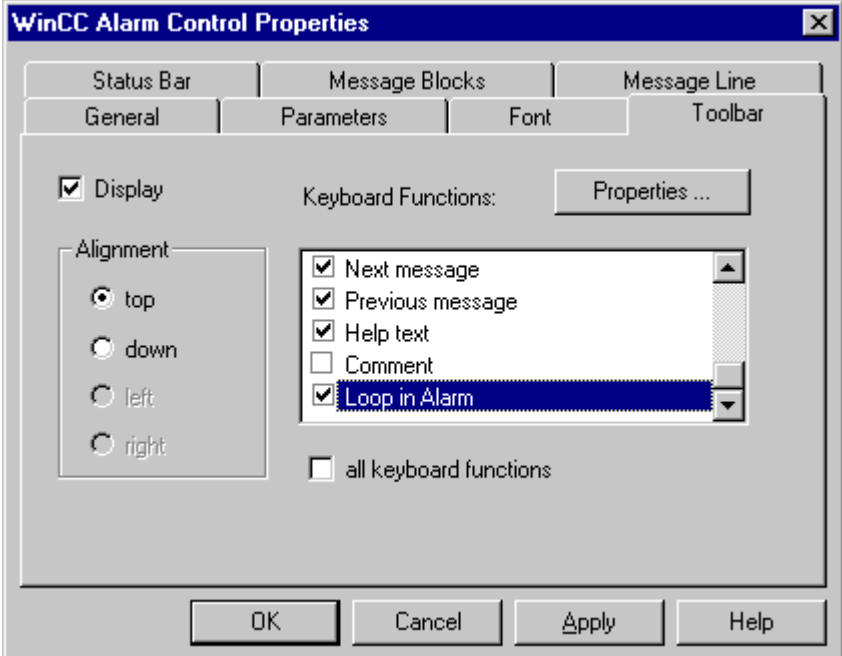
Implementation of the Sample


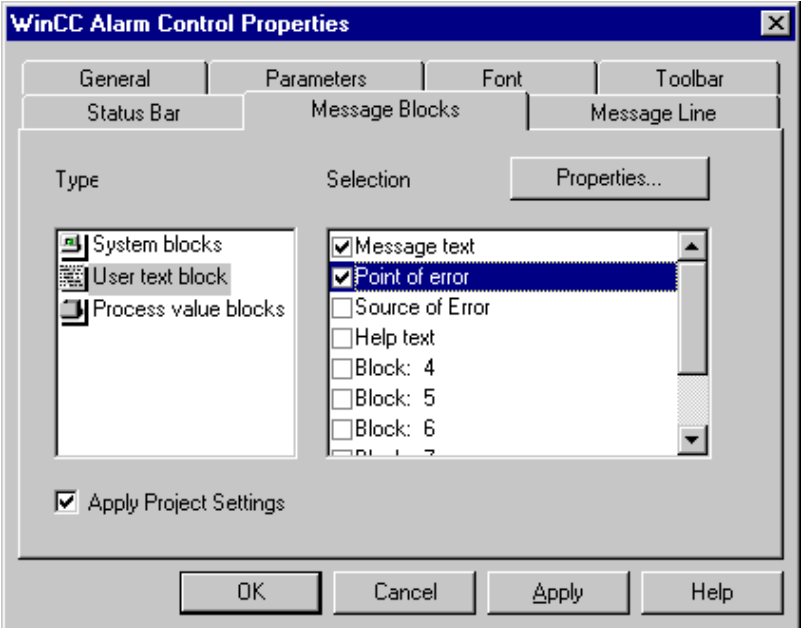
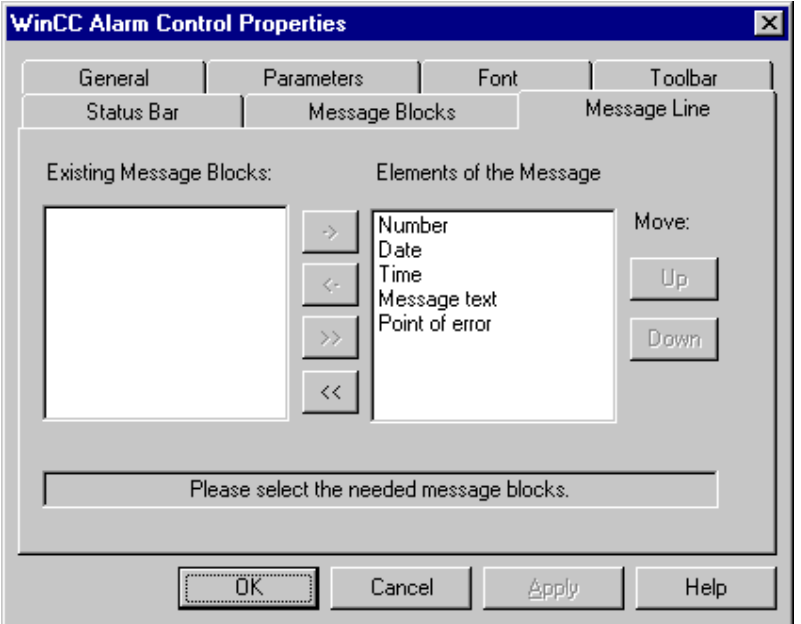
Step	Procedure: Implementation of the Sample																																													
1	Open the <i>Alarm Logging</i> editor from the <i>WinCC Explorer</i> .																																													
2	For each single message, <i>Loop in Alarm</i> must be set. This function makes it possible to perform a direct picture change to the corresponding picture of a message. As the function performing the picture change, <i>OpenPicture</i> is set by default. For this sample, however, a separate function must be created that can perform a picture change in a picture window. The call parameters of this function are predefined by <i>Alarm Logging</i> . In this sample, the <i>ALGLoopInAlarm</i> function has been created in the <i>Global Script</i> editor.																																													
3	In the table window of <i>Alarm Logging</i> ,  on the <i>Loop in Alarm</i> column to open the <i>Loop in Alarm</i> dialog of the selected single message. <table border="1" data-bbox="527 1480 1234 1753"> <thead> <tr> <th></th> <th>Acknowledgement bit</th> <th>Loop in Alarm</th> <th>Group</th> <th>Fi</th> </tr> </thead> <tbody> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>Not set</td> <td></td> <td></td> </tr> </tbody> </table>		Acknowledgement bit	Loop in Alarm	Group	Fi		0	Not set				0	Not set				0	Not set				0	Not set				0	Not set				0	Not set				0	Not set				0	Not set		
	Acknowledgement bit	Loop in Alarm	Group	Fi																																										
	0	Not set																																												
	0	Not set																																												
	0	Not set																																												
	0	Not set																																												
	0	Not set																																												
	0	Not set																																												
	0	Not set																																												
	0	Not set																																												
4	As the <i>Function Name</i> , the function <i>ALGLoopInAlarm</i> is used. For messages that refer to the motors of the first sample, the picture <i>ex_3_chapter_02.pdl</i> is used as the <i>Picture Name/Call Parameter</i> , for messages of the second sample, the picture <i>ex_3_chapter_02a.pdl</i> is used.																																													

Step	Procedure: Implementation of the Sample
	<div data-bbox="483 289 1198 961" style="border: 1px solid gray; padding: 5px;"> <p>Loop In Alarm</p> <p>You can use Loop In Alarm to select a picture that shows the part of the process in which the message occurred.</p> <p>To do this, you normally use the "Open Picture" function.</p> <p>If you want to use a different function to select the picture, make sure that this other function also uses a string as call parameter.</p> <p>Picture Name / Parameter: <input type="text" value="ex_3_chapter_02.pdl"/>  <input type="button" value="R"/></p> <p>Function Name: <input type="text" value="ALGLoopInAlarm"/> <input type="button" value="..."/> <input type="button" value="R"/></p> <p>Select your Loop-In-Alarm picture. </p> <p style="text-align: right;"><input type="button" value="OK"/> <input type="button" value="Cancel"/></p> </div>
5	<p>The configuration of the <i>Loop in Alarm</i> function can also be performed in the <i>Tag/Action</i> tab of the single message's properties dialog at the <i>Loop in Alarm</i> field.</p> <p>The configurations made in <i>Alarm Logging</i> are saved.</p>

Implementation in the Graphics Designer

Step	Procedure:
1	Open the <i>Graphics Designer</i> and create a new picture. In this sample, this is the <i>ex_3_chapter_02b.pdl</i> picture.
2	To display the messages configured in <i>Alarm Logging</i> , a <i>WinCC Alarm Control</i> is used. It is selected from the Object Palette's <i>Control</i> selection menu and then placed in the picture.
3	<p>After placing the Control in the picture, its <i>configuration dialog</i> will be displayed automatically.</p> <p>The <i>configuration dialog</i> can be exited by clicking on <i>OK</i>.</p> <p>Open the Control's <i>Properties</i> dialog. This dialog is displayed via a  on the Control.</p> <p>In the <i>General Information</i> tab, all settings can be made. It is not necessary to make a selection, since all single messages appearing are to be displayed.</p>

Step	Procedure:
4	<p>In the <i>Toolbar</i> tab, the following check-boxes are selected:</p> <ul style="list-style-type: none"> • Single Acknowledgment • Group Acknowledgment • Auto-Scroll On/Off • Report Functions • Beginning of the List • End of List • Next Message • Previous Message • Infotext • Loop in Alarm 

Step	Procedure:
5	<p>In the <i>Message Blocks</i> tab, the columns are selected that will later be displayed in the message line. In this sample, <i>system blocks</i> are selected in the <i>Type</i> field using the . In the right window, <i>Date</i>, <i>Time</i> and <i>Number</i> are selected. For the <i>User Text Blocks</i> entry, <i>Message Text</i> and <i>Point of Error</i> are selected.</p> 
6	<p>In the <i>Message Line</i> tab, the previously selected <i>Message Blocks</i> are assigned to the message line. The <i>Available Message Blocks</i> fields lists the available columns. By pressing on the \rightarrow button, each message block can individually be added to the message line. Via the \gg button, all message blocks listed in the window can be assigned to the message line at one time. The properties dialog is exited by clicking on <i>OK</i>.</p> 

Project Function ALGLoopInAlarm

```
void ALGLoopInAlarm(char* PictureName)
{
    SetPictureName("ex_0_startpicture_00.pdl", "workspace", PictureName);
}
```

- Call of the *SetPictureName* function to perform the picture change. This function cannot be used directly in *Alarm Logging*, since the number and type of its call parameters does not match with the specified ones.

Note:

In the toolbar of the WinCC Alarm Control, a button for the report functions is provided. The implementation of a message sequence report and its activation is described in the Message Sequence Report (ex_3_chapter_02b.pdl) sample of the *Report Designer* chapter.

Note for the General Application

The following adaptations must be made before the general application:

- The *Loop in Alarm* functions configured for the individual messages must be adapted to meet your requirements.
- The display type of the message window must adapted to meet your requirements.

4.2.5 Message Archiving (ex_3_chapter_02c.pdl)

Task Definition

A message archive is to be created as a short-term archive for 200 messages. All messages are to be displayed in a message window.

The message window is to be controlled by a user-defined toolbar. This toolbar contains two special selection buttons that allow the user to either display messages from sample 1 or sample 2.

Implementation Concept

This sample uses the messages created in the previous samples. In addition, a message archive is configured.

The message window is created in the *Graphics Designer* using a WinCC Alarm Control.

The toolbar is implemented using several *Windows Objects* → *Buttons*, *Smart Objects* → *Status Displays* and *Smart Objects* → *Graphic Objects*.

A project function is needed that makes the selection in the message window if the selection buttons are pressed.


Creation of the Required Tags


Step	Procedure: Creation of the Required Tags
1	A total of three tags of the <i>Binary Tag</i> type are created. In this sample, these are the <i>BINi_ex_alg_00</i> , <i>BINi_ex_alg_01</i> and <i>BINi_ex_alg_02</i> tags.

Note:





The configurations made in the first and second sample are considered as complete. They will not be explained separately again, however, this sample is based on them.

Implementation in Alarm Logging

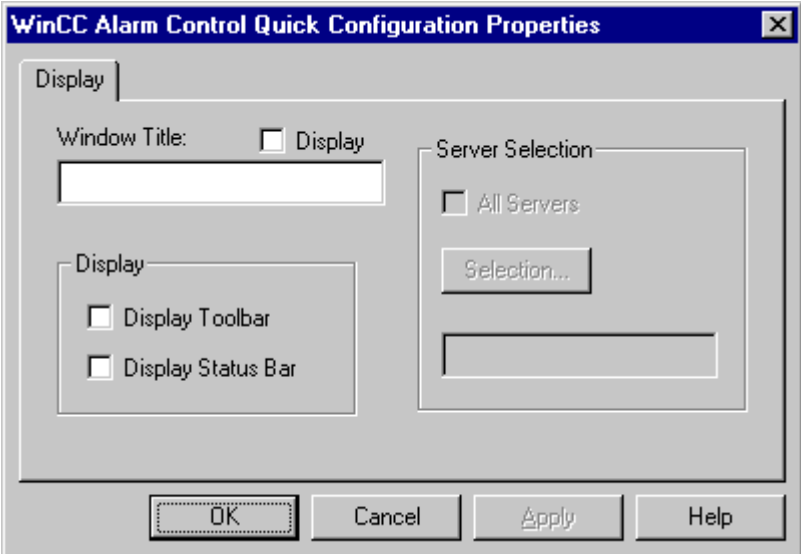

Step	Procedure: Implementation in Alarm Logging
1	Open the <i>Alarm Logging</i> editor from the <i>WinCC Explorer</i> .
2	Via a  on the <i>Archives</i> entry, the <i>Archive Parameters</i> dialog is opened.


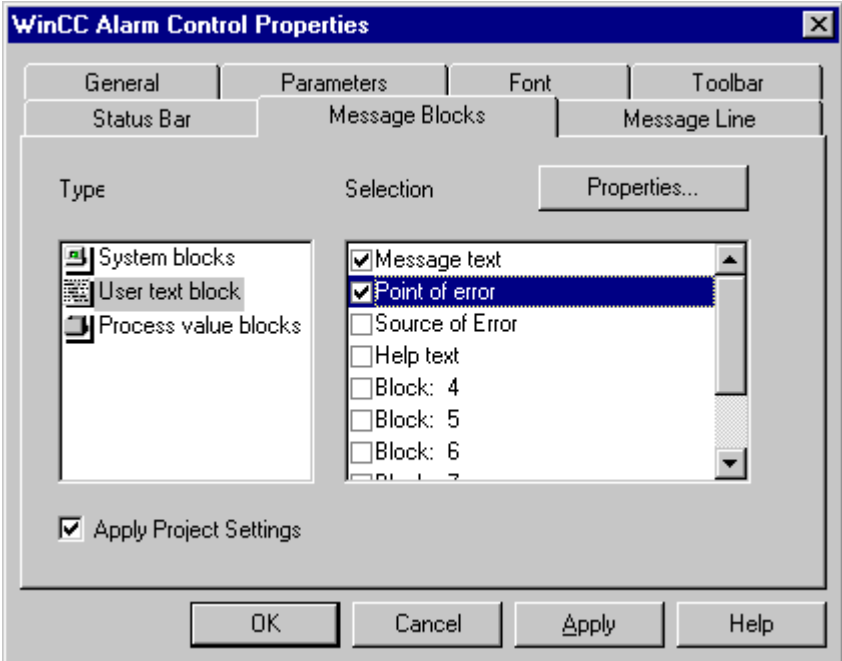





The screenshot shows a tree view with 'Group messages', 'Archives', and 'Reports'. A right-click context menu is open over 'Archives', showing 'Add/Remove...' and 'Properties'.




Step	Procedure: Implementation in Alarm Logging
3	<p>In this dialog, the short-term archive check-box is selected.</p> 
4	<p>In the right window, the icon for the short-term archive will be displayed. Via a  on this icon, the properties dialog of the short-term archive is opened.</p> 
5	<p>The archive is to be stored on disk. In the <i>Number of Entries</i> entry field, specify 200. A <i>Selection</i> is not performed.</p> 

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Open the <i>Graphics Designer</i> and create a new picture. In this sample, this is the <i>ex_3_chapter_02c.pdl</i> picture.
2	<p>After placing the Control in the picture, its <i>configuration dialog</i> will be displayed automatically.</p> <p>As the <i>Window Title</i>, <i>MessageWindow_04</i> is entered. The <i>Display</i> check-box remains deselected. In the C-Actions created later, this window title is used to reference the corresponding Control.</p> <p>The <i>Toolbar</i> and <i>Status Bar</i> check-boxes are deselected.</p> <p>The <i>configuration dialog</i> can be exited by clicking on <i>OK</i>.</p> 
3	<p>Open the Control's <i>Properties</i> dialog. This dialog is displayed via a  on the Control.</p> <p>In the <i>General Information</i> tab, all settings can be made. It is not necessary to make a selection, since all single messages appearing are to be displayed.</p>

Step	Procedure: Implementation in the Graphics Designer
4	<p>In the <i>Message Blocks</i> tab, the columns are selected that will later be displayed in the message line. In this sample, <i>system blocks</i> are selected in the <i>Type</i> field using the . In the right window, <i>Date</i>, <i>Time</i> and <i>Number</i> are selected. For the <i>User Text Blocks</i> entry, <i>Message Text</i> and <i>Point of Error</i> are selected.</p>  <p>The screenshot shows the 'WinCC Alarm Control Properties' dialog box with the 'Message Blocks' tab selected. The 'Type' list on the left contains 'System blocks', 'User text block', and 'Process value blocks', with 'System blocks' selected. The 'Selection' list on the right contains 'Message text', 'Point of error', 'Source of Error', 'Help text', 'Block: 4', 'Block: 5', and 'Block: 6', with 'Message text' and 'Point of error' checked. The 'Apply Project Settings' checkbox is also checked. Buttons for 'OK', 'Cancel', 'Apply', and 'Help' are at the bottom.</p>

Step	Procedure: Implementation in the Graphics Designer
5	<p>In the <i>Message Line</i> tab, the previously selected <i>Message Blocks</i> are assigned to the message line. The <i>Available Message Blocks</i> fields lists the available columns. By pressing on the -> button, each message block can individually be added to the message line. Via the >> button, all message blocks listed in the window can be assigned to the message line at one time. The properties dialog is exited by clicking on <i>OK</i>.</p> 
6	<p>For the toolbar, several <i>Windows Objects</i> → <i>Buttons</i> are configured that simulate the pressing of the individual buttons via special standard functions.</p>
7	<p>A <i>Button</i> for calling the selection dialog and a <i>Button</i> for calling the Infotext dialog are configured. The associated standard functions are: <code>ACX_OnBtnInfo()</code> <code>ACX_OnBtnSelect()</code></p> 
8	<p>As the replacement for the button that turns the auto-scroll function on and off, a <i>Smart Object</i> → <i>Status Display</i> is used. In this sample, the <i>Status Display3</i> object is used.</p> <p>At <i>Properties</i> → <i>State</i> → <i>Current Status</i>, a <i>Tag Connection</i> to the <i>BINi_ex_alg_00</i> tag is created. This tag contains the information whether auto-scrolling is turned on or off. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, a <i>C-Action</i> is created that inverts the status of the <i>BINi_ex_alg_00</i> tag and calls the standard function <code>ACX_OnBtnScroll()</code>. At the opening of the picture, the <i>BINi_ex_alg_00</i> tag is set to 0, since auto-scroll is turned if the message window is reselected.</p> 

Step	Procedure: Implementation in the Graphics Designer
9	<p>If auto-scroll is turned off, the navigation in the message window is to be performed via four special buttons. These buttons replace the corresponding buttons on the standard toolbar with the following standard functions: <code>ACX_OnBtnMsgFirst()</code><code>ACX_OnBtnMsgLast()</code><code>ACX_OnBtnMsgNext()</code><code>ACX_OnBtnMsgPrev()</code></p> <p>These buttons are made inoperational via a <i>Smart Object</i> → <i>Graphic Object</i> which is placed over them if auto-scroll is turned on. This is accomplished via a <i>Tag Connection</i> to the <code>BINi_ex_alg_00</code> tag at <i>Properties</i> → <i>Miscellaneous</i> → <i>Display</i>.</p> 
10	<p>Via two <i>Smart Objects</i> → <i>Status Displays</i>, the switching between the display types <i>Message Window</i> and <i>Short-Term Archive Window</i> is to be realized. The current status of the message window is stored in the <code>BINi_ex_alg_01</code> tag that must be set to zero at the opening of the picture, since the message window is displayed as a <i>Short-Term Archive Window</i> when reopened.</p> <p>For <i>Status Display1</i>, create a <i>Tag Connection</i> at <i>Properties</i> → <i>State</i> → <i>Current Status</i> to the tag <code>BINi_ex_alg_01</code>. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Operator-Control Enable</i>, create a <i>Dynamic Dialog</i> that makes the object operational only if the message window displays the short-term archive, i.e. the <code>BINi_ex_alg_01</code> tag has the status 0. At <i>Events</i> → <i>Mouse</i> → <i>Press Left</i>, create a <i>C-Action</i> that simulates the pressing of the corresponding button on the toolbar and inverts the <code>BINi_ex_alg_01</code> tag. The <i>Status Display2</i> object is configured in the same way. The following standard functions are used: <code>ACX_OnBtnMsgWin()</code><code>ACX_OnBtnArcShortt()</code></p> 
11	<p>Via two additional <i>Windows Objects</i> → <i>Buttons</i>, direct selections are made in the message window. A selection can be made to view the messages referring to the motors or to the containers. The selection is performed by a project function created in the <i>Global Script</i> editor. The message numbers, in between which the displayed messages are lying, are transferred to this function. In the sample, this function is called <code>SetMsgNrSelection</code>.</p> 

Project Function for Setting a Selection

```

BOOL SetMsgNrSelection(DWORD dwFrom, DWORD dwTo, LPSTR MsgTem)
{
    PCMN_ERROR      pError;
    BOOL            fRet;
    MSG_FILTER_STRUCT Filter;

    memset(&Filter, \0, sizeof( MSG_FILTER_STRUCT ) );
    strcpy( Filter.szFilterName, MsgTem);
    Filter.dwFilter = MSG_FILTER_NR_FROM|MSG_FILTER_NR_TO;
    Filter.dwMsgNr[0] = dwFrom;
    Filter.dwMsgNr[1] = dwTo;

    fRet = MSRTSetMsgWinFilter( &Filter, pError );

    if (fRet == FALSE)
    {
        printf("Error MSRTSetMsgWinFilter\r\n" );
        return FALSE;
    }
    else
        return TRUE;
}

```

- Reservation of memory for the created *Filter* filter structure.
- Assignment of the values to the structure member of the filter structure relevant for this application. As the *szFilterName*, the name of the message window template, to which the filter is referring, must be used. In the *dwMsgNr* array, the start and end values of the message numbers to be selected are entered. These values are supplied as transfer parameters while the function is called. The *dwFilter* switch is set in such a way that it identifies the filter structure as a number filter.
- Call of the API function *MSRTSetMsgWinFilter*, which applies the created filter to the selected message window template.


Note for the General Application

The following adaptations must be made before the general application:

- The display type of the message window must adapted to meet your requirements.
- The appearance and the toolbar elements must be adapted to meet your requirements.

4.2.6 Group Messages (ex_8_generator_00.pdl)



In runtime, the sample pertaining to this topic is accessed by selecting the *Button* displayed above using the . Via the check-box labeled *active* in this picture, the message generator can be turned on. It creates different messages in 10 second intervals.

Task Definition

In a picture, warnings are to be displayed that inform about the presence of a certain type of message.

These messages were already configured in the *Bit Message Procedure* (ex_3_chapter_02.pdl) and *Limit Value Monitoring (Continuation)* samples and are applied to this sample. Pending warnings and alarms in the container picture and errors occurring in the motor picture are to be pointed out. The alarm has priority before the failure and the error. Via a button, a jump to the corresponding picture is made if a message is pending.

Implementation Concept

The single messages to be monitored are combined into a group message. If a single message is generated, the group message is generated as well. A status tag and a status bit are assigned to this group message. Via a *Smart Object* → *Status Display*, the current status of this tag is evaluated and an appropriate symbol displayed.

Note:

The configurations made in the first and second sample are considered as complete. They will not be explained separately again, however, this sample is based on them.

Creation of the Required Tags

Step	Procedure: Creation of the Required Tags
1	Creation of a total of three tags of the <i>Unsigned 16-Bit Value</i> type in Tag Management. In this sample, these are the <i>U16i_ex_alg_20</i> , <i>U16i_ex_alg_21</i> and <i>U16i_ex_alg_22</i> tags. They serve as status, lock and acknowledge tags.





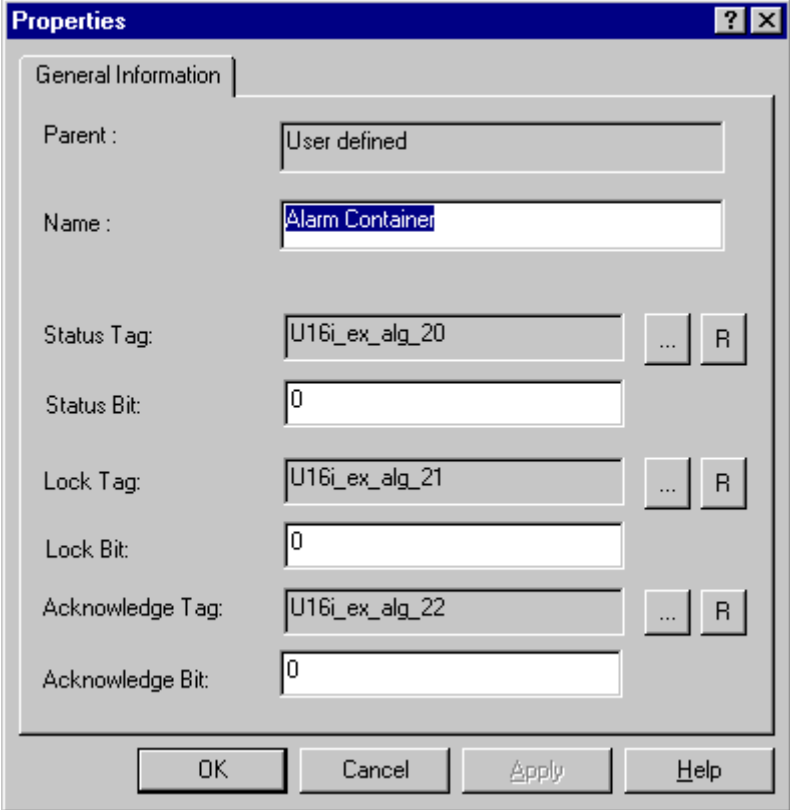

General Information




If a new message class is created, a group message is automatically created for this message class as well. All messages within this message class are transferred to the group message. The properties of the message classes and message types in the group message can be changed independently and therefore be connected to different status, lock and acknowledge tags.

In this sample, however, it is assumed that other pictures using the same message classes exist in the project. This means that the automatically generated group messages cannot be used, since the corresponding group message must also identify the picture in which it occurred.


This means that user-defined group messages must be created.




Creation of new Group Messages


Step	Procedure: Creation of new Group Messages
1	<p>Open the <i>Alarm Logging</i> editor.</p> <p> Click on the <i>Group Messages</i> entry to expand it - two sub-entries will be displayed. These are the <i>Message Class</i> and <i>User-Defined</i> points.</p> <p> Via a  on the <i>User-Defined</i> entry, the <i>New Group Message</i> dialog is accessed.</p> 
2	<p>In the displayed dialog, <i>Alarm Container</i> is entered as the <i>Name</i>. As the <i>Status</i>, <i>Lock</i> and <i>Acknowledge Tags</i>, the previously created tags are set. As the bit number, <i>0</i> is used in each case.</p> <p>Exit the dialog box by clicking on <i>OK</i>.</p> 
3	<p>In the same manner, two additional group messages are created. They use the same <i>Status</i>, <i>Lock</i> and <i>Acknowledge Tags</i>, but the bit numbers <i>1</i> and <i>2</i> respectively. In the right window, the icons of the newly created group messages are displayed.</p> 

Step	Procedure: Creation of new Group Messages
4	<p>Via a  on one of these icons, the <i>New Single Message(s)</i> dialog can be opened. For each group message, the message numbers of the corresponding single messages are entered and the dialog is closed by clicking on <i>OK</i>.</p> <div data-bbox="526 405 1230 779" style="border: 1px solid gray; padding: 5px;"> <p>New Single Message(s)</p> <p>Message Number(s)</p> <p><input type="text" value="13-18"/> Separate single messages by commas or indicate a range. For example: 1,2,3,5-10</p> <p><input type="checkbox"/> Only then insert the single message(s), if it does not already belong to a group.</p> <p style="text-align: right;"> <input type="button" value="OK"/> <input type="button" value="Cancel"/> </p> </div>
5	<p>Via a  on the <i>User-Defined</i> entry in the navigation window, the individual group messages are displayed as sub-entries. If one of these entries is selected with the , the right window will display the icons of all added single messages.</p> <div data-bbox="526 951 857 1140" style="border: 1px solid gray; padding: 5px;"> <ul style="list-style-type: none"> <input type="checkbox"/> Group messages <ul style="list-style-type: none"> <input type="checkbox"/> Message Class <input type="checkbox"/> User defined <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Alarm Container <input type="checkbox"/> Failure Motor <input type="checkbox"/> Warning Container </div>

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	<p>In the <i>Graphics Designer</i>, a new picture is created. In this sample, this is the <i>ex_8_generator_00</i> picture.</p> <p>In this picture, a <i>Smart Object</i> → <i>Status Display</i> is configured, which displays the current status of the group messages. In the sample, this is the <i>Status Display1</i> object. Following the configuration, the status of the group messages is stored by <i>Alarm Logging</i> in the <i>U16i_ex_alg_20</i> tag.</p> <p>For each status, a corresponding bitmap must be designed. This means that bitmaps for three unacknowledged stati, three acknowledged stati and for the status where no message is pending are required. At <i>Properties</i> → <i>State</i> → <i>Current Status</i>, a <i>C-Action</i> is created that controls the status depending on the <i>U16i_ex_alg_20</i> tag and the required priority.</p> <div data-bbox="532 1686 669 1793" style="text-align: center;">  </div>

Step	Procedure: Implementation in the Graphics Designer
2	<p>Additionally, a <i>Windows Object</i> → <i>Button</i> is configured that performs a picture change to the picture from which the message originated if a group message is displayed. In the sample, this is the <i>Button1</i> object.</p> <p>Via a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, the current status of the group message is requested and the corresponding picture change performed. If no message is pending, an additional <i>Windows Object</i> → <i>Button</i> is placed over the just described <i>Button</i>. This button makes the other one inoperational and displays this visually. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Operator-Control Enable</i>, set this button to <i>No</i>.</p> 
3	<p>Configure another <i>Windows Object</i> → <i>Button</i> which is used to acknowledge the currently displayed group message. In the sample, this is the <i>Button3</i> object.</p> <p>Via a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, it is determined if a group message needs to be acknowledged and if yes, which one. If a message needs to be acknowledged, the corresponding bit in the configured acknowledge tag <i>U16i_ex_alg_22</i> is set and then immediately reset. If no unacknowledged message is pending, an additional <i>Windows Object</i> → <i>Button</i> is placed over the just described <i>Button</i> to make it inoperational and to display this visually. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Operator-Control Enable</i>, set this button to <i>No</i>.</p> 
4	<p>Configure another picture; in the sample, this is the <i>ex_8_generator_01</i> picture.</p> <p>In this picture, three <i>Windows Objects</i> → <i>Check-Boxes</i> are configured. In the sample, these are the <i>Check-Box1</i>, <i>Check-Box2</i> and <i>Check-Box3</i> objects.</p> <p>At <i>Events</i> → <i>Property Topics</i> → <i>Output/Input</i> → <i>Selected Boxes</i>, create a <i>C-Action</i> for each check-box that locks or enables its corresponding group message. The respective locks are stored by <i>Alarm Logging</i> in the <i>U16i_ex_alg_21</i> tag according to the configuration. Since a lock can also be set from the other side, a <i>C-Action</i> must be created at <i>Properties</i> → <i>Output/Input</i> → <i>Selected Boxes</i>. This action is triggered upon the change of the <i>U16i_ex_alg_21</i> tag and checks if the status of the lock controlled by the corresponding <i>Check-Box</i> has changed.</p> 

Step	Procedure: Implementation in the Graphics Designer
5	<p>In the initially created picture <i>ex_8_generator_00</i>, a <i>Smart Object</i> → <i>Picture Window</i> is created in which the <i>ex_8_generator_01</i> picture is set at <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>. Set the <i>Property</i> → <i>Miscellaneous</i> → <i>Display to No</i>.</p> <p>Another <i>Windows Object</i> → <i>Button</i> is required that makes the previously configured <i>Picture Window</i> visible via a <i>Direct Connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>.</p> 

C-Action for Determining the current Status

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
WORD state;

state = GetTagWord("U16i_ex_alg_20");

if ((state&1)&&(state&256)) return 6;
else if ((state&2)&&(state&512)) return 5;
else if (state&1) return 3;
else if ((state&4)&&(state&1024)) return 4;
else if (state&2) return 2;
else if (state&4) return 1;
else return 0;
}
```

- Reading of the status tag written by *Alarm Logging*.
- Setting of the current status depending on this tag. If multiple group messages are pending, the defined priority decides which one is displayed. In this sample, the priority is as follows, starting with the highest priority level:
 - Container Alarm
 - Motor Failure
 - Acknowledged Container Alarm
 - Container Warning
 - Acknowledged Motor Failure
 - Acknowledged Container Warning

C-Action for Performing the Picture Change

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    Int value;

    Value = GetIndex(lpszPictureName, "Status Display1");

    If ((value==2)|| (value==5))
        SetPictureName("ex_0_startpicture_00.PDL",
            "workspace", "ex_3_chapter_02.PDL");
    else if (value>0)
        SetPictureName("ex_0_startpicture_00.PDL",
            "workspace", "ex_3_chapter_02a.PDL");
}
```

- Determination of the currently displayed status of the status display.
- Depending on this displayed status, the picture change is performed. If the status is 0, no action will be performed.

C-Action for Acknowledging the Displayed Message

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    WORD state;

    state = GetTagWord("U16i_ex_alg_20");

    if ((state&1)&&(state&256)){
        SetTagWord("U16i_ex_alg_22",
            (WORD)(1|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
            (WORD)(~1&GetTagWord("U16i_ex_alg_22")));
    }
    else if ((state&2)&&(state&512)){
        SetTagWord("U16i_ex_alg_22",
            (WORD)(2|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
            (WORD)(~2&GetTagWord("U16i_ex_alg_22")));
    }
    else if ((state&4)&&(state&1024)){
        SetTagWord("U16i_ex_alg_22",
            (WORD)(4|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
            (WORD)(~4&GetTagWord("U16i_ex_alg_22")));
    }
}
```

- Reading of the current status of the group messages.
- If a message to be acknowledged is pending, it will be acknowledged. If multiple messages to be acknowledged are pending, the one with the highest priority is acknowledged.

C-Action for Setting a Lock

```
#include "opdcfop.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l
{
    DWORD dwServiceID;
    MSG_RTGROUPSET_STRUCT mGroup;
    CMN_ERROR Error;
    BOOL fRet;
    time_t Time;
    struct tm* TimeStruct;

    time(&Time);
    TimeStruct = localtime(&Time);

    mGroup.stTime.wYear=(WORD)(TimeStruct->tm_year+1900);
    mGroup.stTime.wMonth=(WORD)(TimeStruct->tm_mon+1);
    mGroup.stTime.wDay=(WORD)(TimeStruct->tm_mday);
    mGroup.stTime.wHour=(WORD)(TimeStruct->tm_hour);
    mGroup.stTime.wMinute=(WORD)(TimeStruct->tm_min);
    mGroup.stTime.wSecond=(WORD)(TimeStruct->tm_sec+1);

    mGroup.fIDUsed=FALSE;
    strcpy(mGroup.szName, "Alarm Behälter");
    mGroup.dwData=value;

    MSRTStartMsgService(&dwServiceID, NULL, NULL,
                        MSG_NOTIFY_MASK_ALL, (LPVOID)0, &Error);

    fRet=MSRTLockGroup (dwServiceID, &mGroup, &Error);
    if (fRet==FALSE)
        printf("Error in MSRTLockGroup(!!!) %s\r\n", Error.szErrorText);
    else printf("Executed MSRTLockGroup(!!!) \r\n");

    MSRTStopMsgService (dwServiceID, &Error );
}
```

- Definition of the required variables. *mGroup* is a structure that must be transferred to the function responsible for setting the lock.
- Determination of the current system time. This value is transferred to the *stTime* structure member of the SYSTEMTIME type.
- The *fIDUsed* structure member indicates if the desired group message - that is to be locked or enabled - is to be defined via its name of ID. The value FALSE specifies that the group message is identified via its name.
- *szName* contains the name of the desired group message.
- *dwDate* indicates if it is to be set or locked. The current status is transferred to the check-box.
- Starting of a message service via the function *MSRTStartMsgService*.
- Call of the function for locking or enabling the group message *MSRTLockGroup*.
- Ending of the message service via the function *MSRTStopMsgService*.


Note for the General Application

The following adaptations must be made before the general application:

- The single messages combined into a group message must be adapted to meet your own requirements.
- The group message display, the display priority and the picture changes to be performed must be adapted to meet your requirements.

4.3 Report Designer

A rectangular button with rounded corners, a light blue gradient, and a thin border. The text "Report Designer" is centered on the button in a blue, sans-serif font.

In runtime, the samples pertaining to this topic are accessed by selecting the button displayed above using the . The samples are configured in the *ex_3_chapter_03.pdl* picture. Additional samples are spread out throughout the sample project.

General Information

The *Report Designer* is part of the WinCC base package and provides functions for the creation and output of reports. Creation pertains to the creation of the report layout in the configuration system of the *Report Designer* and output to the printing of the report.

Note:

The supplied system layouts can be used directly or be copied and then modified to meet your specifications. The names of the system layouts and the system print jobs always start with the character @.

4.3.1 Picture Documentation (ex_3_chapter_03.pdl)

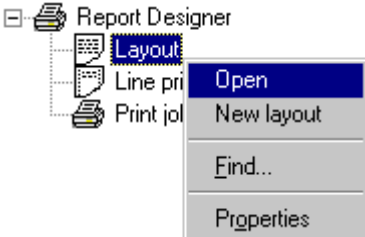
Task Definition


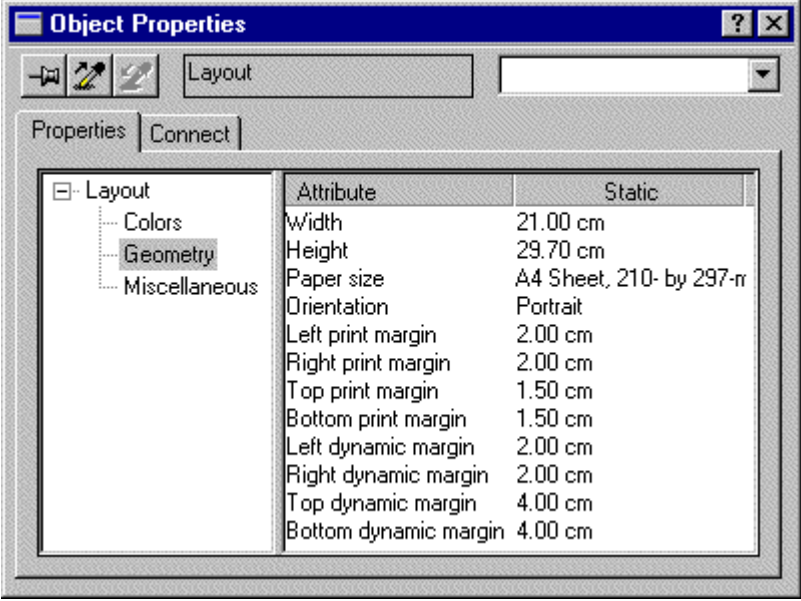

A comprehensive documentation of all pictures contained in a project is to be created. For each picture, the documentation should include a graphical display, general information about the picture, a listing of all objects and a listing of all set picture properties.


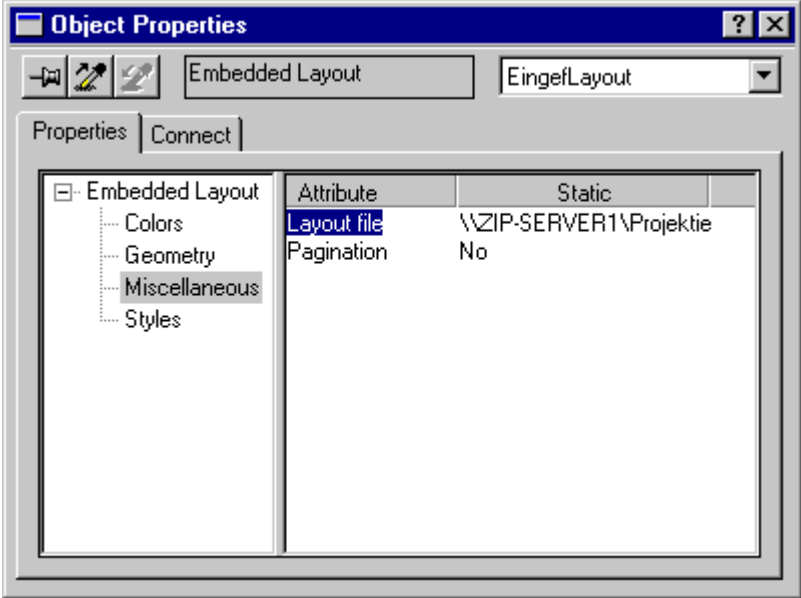
Implementation Concept

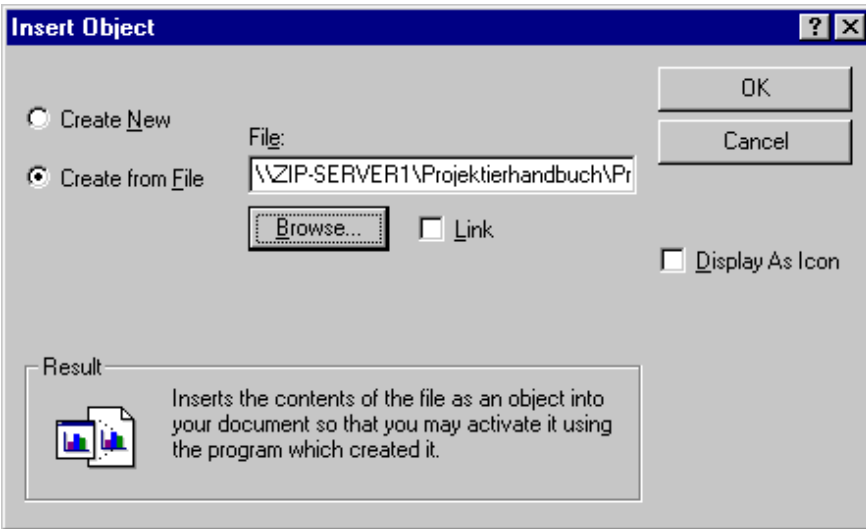

A system layout is available that matches the requirements made. This is the layout @PDLpicture (compact).rpl. This layout is copied and adapted to your needs.

Implementation in the Report Designer


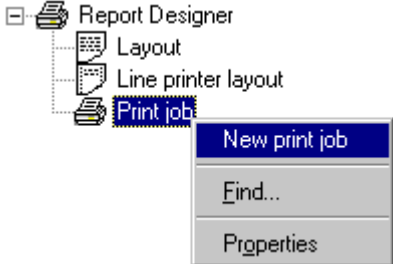


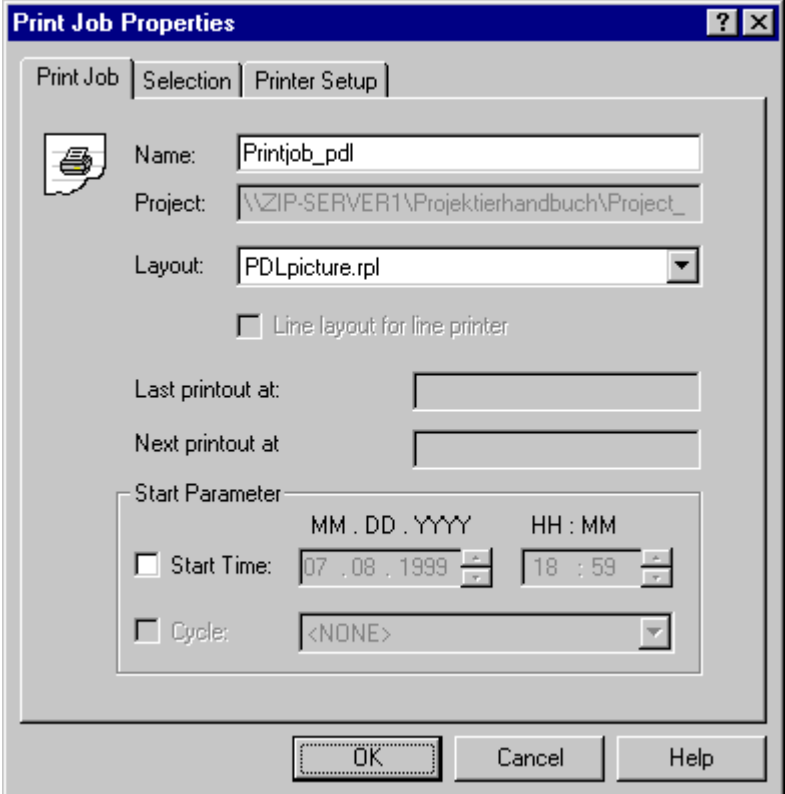
Step	Procedure: Implementation in the Report Designer
1	<p>Open the <i>Report Designer</i> editor from the <i>WinCC Explorer</i>.</p> 
2	<p>Via the <i>File</i> → <i>Open...</i> menus, open the system layout @PDLpic.rpl and save it under a different name via the <i>File</i> → <i>Save As...</i> menus. In this sample, the name <i>PDLpicture.rpl</i> is used.</p>

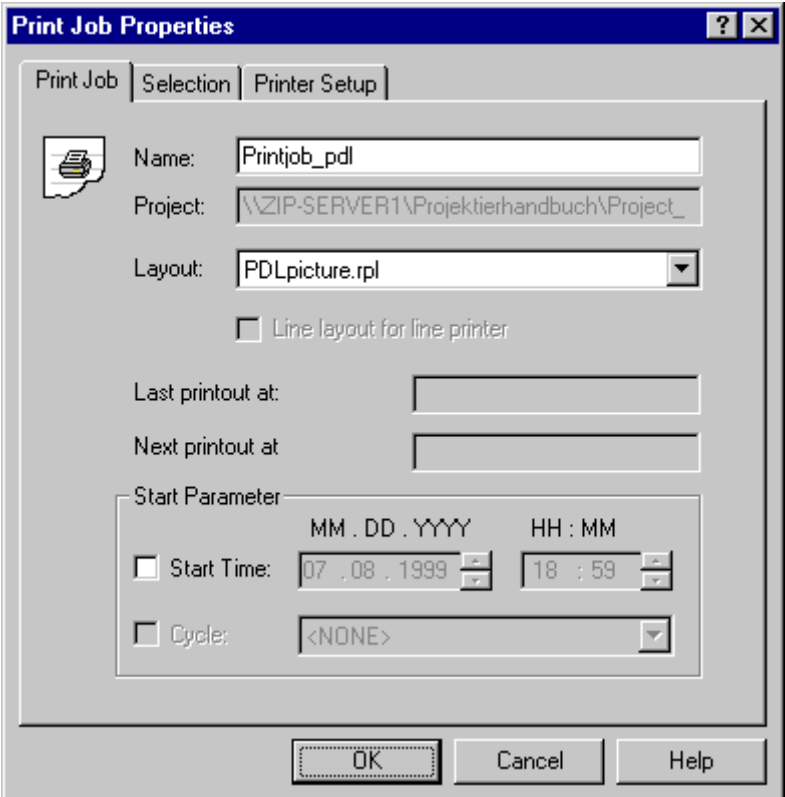
Step	Procedure: Implementation in the Report Designer																										
3	<p>Via a  on the blank space of the layout, its properties dialog is opened.</p> <p>In the <i>Properties</i> tab, the general geometric settings can be made at the <i>Geometry</i> property.</p> <p>At the <i>Miscellaneous</i> property, you can specify a cover sheet and a final page for the report. For this sample, a cover sheet is specified.</p>  <table border="1" data-bbox="516 653 1235 1039"> <thead> <tr> <th>Attribute</th> <th>Static</th> </tr> </thead> <tbody> <tr> <td>Width</td> <td>21.00 cm</td> </tr> <tr> <td>Height</td> <td>29.70 cm</td> </tr> <tr> <td>Paper size</td> <td>A4 Sheet, 210- by 297-r</td> </tr> <tr> <td>Orientation</td> <td>Portrait</td> </tr> <tr> <td>Left print margin</td> <td>2.00 cm</td> </tr> <tr> <td>Right print margin</td> <td>2.00 cm</td> </tr> <tr> <td>Top print margin</td> <td>1.50 cm</td> </tr> <tr> <td>Bottom print margin</td> <td>1.50 cm</td> </tr> <tr> <td>Left dynamic margin</td> <td>2.00 cm</td> </tr> <tr> <td>Right dynamic margin</td> <td>2.00 cm</td> </tr> <tr> <td>Top dynamic margin</td> <td>4.00 cm</td> </tr> <tr> <td>Bottom dynamic margin</td> <td>4.00 cm</td> </tr> </tbody> </table>	Attribute	Static	Width	21.00 cm	Height	29.70 cm	Paper size	A4 Sheet, 210- by 297-r	Orientation	Portrait	Left print margin	2.00 cm	Right print margin	2.00 cm	Top print margin	1.50 cm	Bottom print margin	1.50 cm	Left dynamic margin	2.00 cm	Right dynamic margin	2.00 cm	Top dynamic margin	4.00 cm	Bottom dynamic margin	4.00 cm
Attribute	Static																										
Width	21.00 cm																										
Height	29.70 cm																										
Paper size	A4 Sheet, 210- by 297-r																										
Orientation	Portrait																										
Left print margin	2.00 cm																										
Right print margin	2.00 cm																										
Top print margin	1.50 cm																										
Bottom print margin	1.50 cm																										
Left dynamic margin	2.00 cm																										
Right dynamic margin	2.00 cm																										
Top dynamic margin	4.00 cm																										
Bottom dynamic margin	4.00 cm																										
4	<p>Via the toolbar buttons displayed below, the static and dynamic parts of the report can be edited.</p> 																										

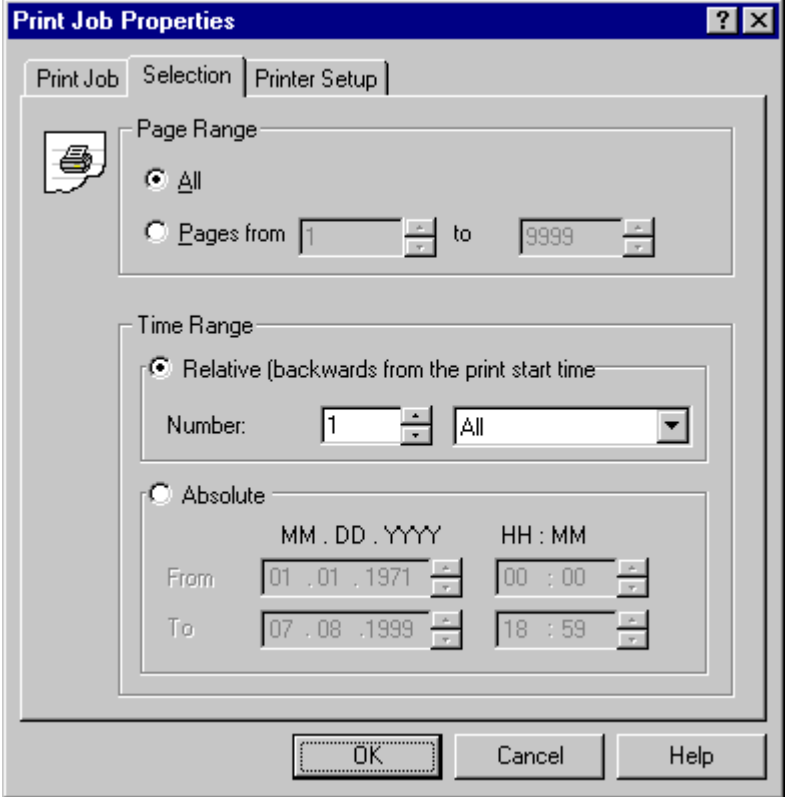
Step	Procedure: Implementation in the Report Designer						
5	<p>The dynamic part of the report contains a <i>Dynamic Object</i> → <i>Embedded Layout</i>. In the sample, this is the <i>EmbedLayout</i> object, which must be changed.</p> <p>Via a  on the dynamic part of the report, select the <i>@PDL picture (compact).rpl</i> layout at <i>Properties</i> → <i>Miscellaneous</i> → <i>Layout File</i>.</p> <p>This layout can be opened and its elements be adapted to meet your requirements. It is, however, recommended, to copy this layout first and then to make changes in the copy. If this is done, the newly created layout must be set in the initial layout at the <i>EmbedLayout</i> object via <i>Properties</i> → <i>Miscellaneous</i> → <i>Layout File</i>.</p>  <table border="1" data-bbox="565 781 1286 1165"> <thead> <tr> <th data-bbox="565 781 808 814">Attribute</th> <th data-bbox="808 781 1286 814">Static</th> </tr> </thead> <tbody> <tr> <td data-bbox="565 814 808 848">Layout file</td> <td data-bbox="808 814 1286 848">\\ZIP-SERVER1\Projektie</td> </tr> <tr> <td data-bbox="565 848 808 882">Pagination</td> <td data-bbox="808 848 1286 882">No</td> </tr> </tbody> </table>	Attribute	Static	Layout file	\\ZIP-SERVER1\Projektie	Pagination	No
Attribute	Static						
Layout file	\\ZIP-SERVER1\Projektie						
Pagination	No						

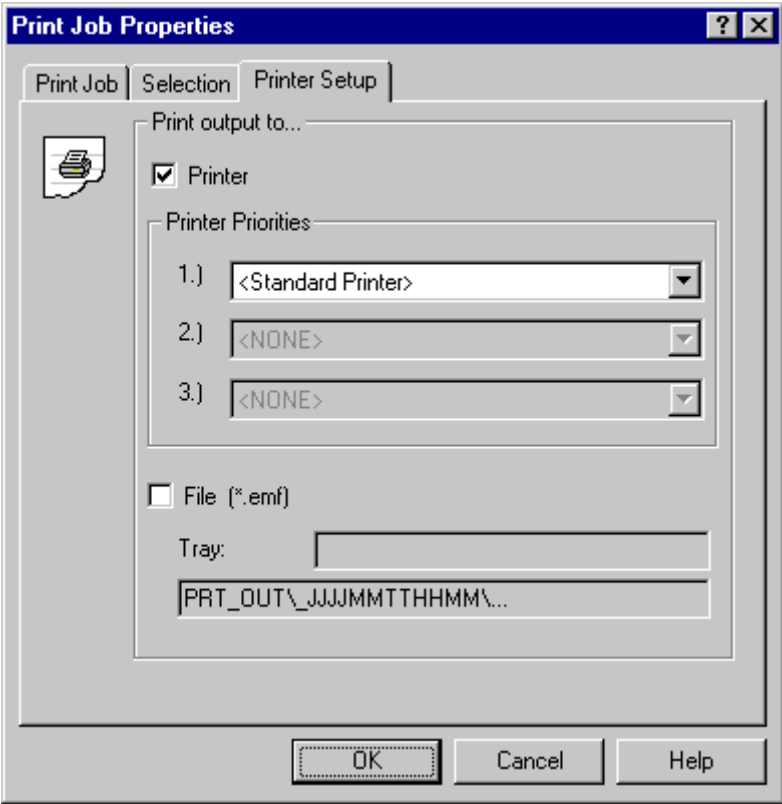

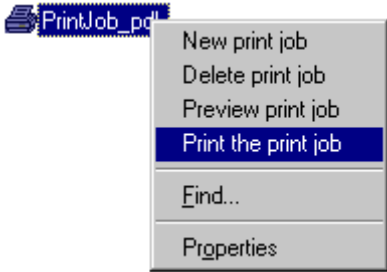
Step	Procedure: Implementation in the Report Designer
6	<p>The static part of the report contains a header and a footer.</p> <p>The footer contains the <i>System Objects Date/Time, Page Number, Project Name</i> and <i>Layout Name</i>.</p> <p>The header contains two <i>Static Objects</i> → <i>Static Texts</i> as well as a <i>System Object</i> → <i>Project Name</i>. Additionally, a logo is displayed using a <i>Static Object</i> → <i>OLE Element</i>. In this sample, the text of the <i>StatText1</i> object is changed to <i>Picture Documentation</i>. To display your own logo, the existing <i>OLEElement1</i> object is deleted. A new <i>Static Object</i> → <i>OLE Element</i> is now configured. From the <i>Insert Object</i> dialog, which is displayed after placing the object in the report, select the <i>Create from File</i> radio-button and specify the bitmap file that contains the logo. The dialog is closed by clicking on <i>OK</i>.</p> 
7	<p>Via the toolbar buttons displayed below, you can switch among the cover sheet, report contents and final page.</p> <p>In this sample, the cover sheet contains two <i>Static Objects</i> → <i>Static Texts</i>, a <i>System Object</i> → <i>Project Name</i> and a <i>Static Object</i> → <i>Static Metafile</i>.</p> 
8	<p>The changes made in the Report Designer are saved and the Report Designer editor is exited.</p>

Creation of a Print Job

Step	Procedure: Creation of a Print Job
1	<p>In the <i>WinCC Explorer</i>, a new print job is created via a .</p> 
2	<p>In the right window, this new print job, with the default name <i>Print Job001</i>, will be added to the existing print jobs. Via a  or a  on this print job, its properties dialog is opened.</p> 

Step	Procedure: Creation of a Print Job
3	<p>In the <i>Print Job</i> tab, the default name is replaced with <i>Printjob_pdl</i>. As the <i>Layout</i>, the previously created <i>PDLpicture.rpl</i> layout is specified.</p> 

Step	Procedure: Creation of a Print Job
4	<p>In the <i>Selection</i> tab, you can specify what to print. In the <i>Page Range</i> field, the <i>All</i> radio-button is selected. A <i>Time Range</i> set will have no influence in this sample.</p>  <p>The screenshot shows the 'Print Job Properties' dialog box with the 'Selection' tab active. It contains three main sections: 'Page Range', 'Time Range', and 'Absolute'. In the 'Page Range' section, the 'All' radio button is selected. In the 'Time Range' section, the 'Relative (backwards from the print start time)' radio button is selected, with the 'Number' set to 1 and 'All' selected in the dropdown menu. The 'Absolute' section shows 'From' as 01.01.1971 00:00 and 'To' as 07.08.1999 18:59. The 'OK' button is highlighted with a dashed border.</p>

Step	Procedure: Creation of a Print Job
5	<p>In the <i>Printer Setup</i> tab, the printer to be used is specified. The data can also be printed to a file.</p> 
6	<p>Via , the print job can be started from the <i>WinCC Explorer</i>. In the same way, a print preview can be displayed.</p> 
7	<p>In the sample project, a preview of the print job can be activated via a <i>Windows Object</i> → <i>Button</i>. This is the <i>Button13</i> object in the <i>ex_3_chapter_03.pdl</i> picture.</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, create a <i>C-Action</i> that starts a preview of the corresponding print job.</p>

C-Action for Starting a Print Job

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    RPTJobPreview("PrintJob_pdl");
}
```

- Call of the *RPTJobPreview* standard function. As the transfer parameter, the name of the print job is used.

Note:

If the layout embedded in the report layout is open in the *Report Designer* editor, the preview and the print job cannot be started.

Note for the General Application

The following adaptations must be made before the general application:

- The layout created can be used without changes. The logo and information displayed by the report must be adapted. These changes are performed as described at *Implementation in the Report Designer*, steps 3 and 4.

4.3.2 Reporting of the WinCC Explorer (ex_3_chapter_03.pdf)


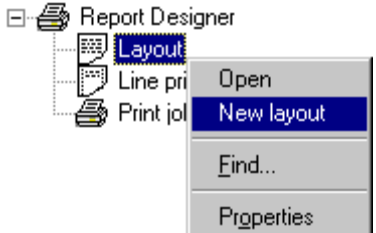

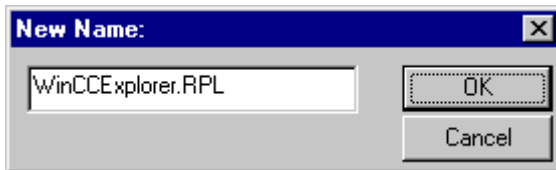

Task Definition

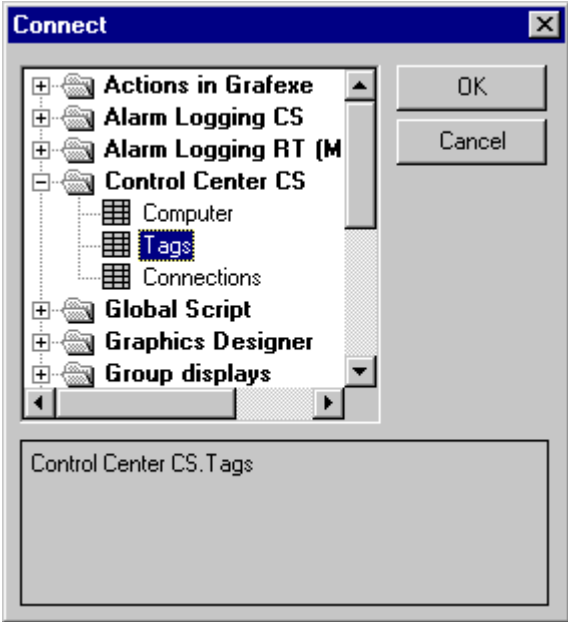






All tags of certain tag groups used in a project are to be documented. The documentation should include the tag name, tag type, tag group, tag parameters and information pertaining to the tag's process connection.

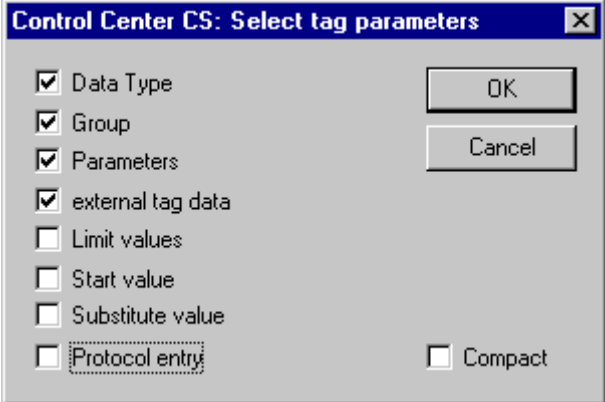
Implementation Concept

A separate layout is configured in the *Report Designer* editor. This layout is not based on any existing layout.

Implementation in the Report Designer

Step	Procedure: Implementation in the Report Designer
1	<p>Creation of a new layout via a  on the <i>Page Layout</i> entry in the <i>WinCC Explorer</i>.</p> 
2	<p>In the right window, this new layout, with the default name <i>NewRPL00.RPL</i>, will be added to the existing page layouts. Via a , this layout can be renamed using a more meaningful name. In this sample, the name <i>WinCC Explorer.rpl</i> is used.</p> 
3	<p>Open the new layout in the <i>Report Designer</i>.</p> <p>In the properties dialog of the layout, which is accessed via a  on the blank space of the report page, at the <i>Miscellaneous</i> property, a <i>Cover Sheet</i> is specified. The remaining settings are kept.</p>
4	<p>In the static part of the report, various <i>Static Objects</i> and <i>System Objects</i> are configured for the header and the footer.</p> <p>The design of the cover sheet is only a suggestion for your own design.</p>

Step	Procedure: Implementation in the Report Designer
5	<p>In the dynamic part of the report, a <i>Dynamic Object</i> → <i>Dynamic Table</i> is configured. In the sample, this is the <i>DynTable1</i> object.</p> <p>After the object has been placed in the report, the <i>Connect</i> dialog will be displayed. From the <i>WinCC Explorer</i> folder, select the <i>Tag</i> entry. Close the dialog box by clicking on <i>OK</i>.</p> 
6	<p>In the <i>Connect</i> tab of the table's properties dialog, several selection options are available.</p> <ul style="list-style-type: none">  Tag parameter selection  Tag group selection  Tag selection  Format
7	<p>Via a  on one of these entries, the corresponding dialog for the data selection is accessed. If a selection is made, it is symbolized by a red check-mark .</p>

Step	Procedure: Implementation in the Report Designer
8	<p>In the tag parameters selection dialog, the check-boxes <i>Data type</i>, <i>Group</i>, <i>Parameters</i> and <i>External Tag Data</i> are selected. Additionally, the check-box <i>Compact</i> is selected. It has the effect that all tag data is displayed in one line.</p> 
9	<p>For the tag group selection, the groups <i>AlarmLogging1</i> and <i>AlarmLogging2</i> are selected in this sample. This selection is only possible, if the check-box <i>All Tag Groups</i> has not been selected.</p> <p>To apply the selection of the tag groups, the check-box <i>All Tags</i> in the tag selection dialog must be deselected.</p> <p>The settings made in the <i>Report Designer</i> are saved.</p>

Note:

The procedure for creating a new print job and for its activation from the *WinCC Explorer* and runtime is detailed in the first sample of the *Report Designer* chapter at *Creation of a Print Job*. The settings can be made in the same manner.

Note for the General Application

The following adaptations must be made before the general application:

- The layout created can be used without changes.

4.3.3 Reporting of Tag Logging CS (ex_3_chapter_03.pdl)




Task Definition

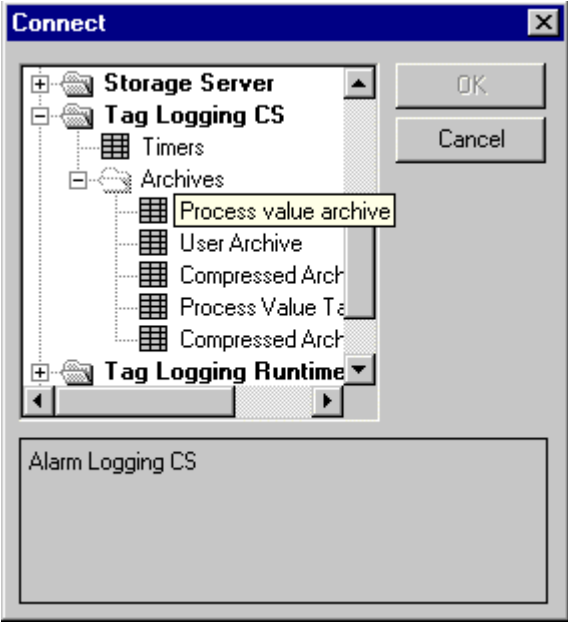

The configuration data of all process value archives created in a project are to be documented. The documentation should contain the general archive data as well as the configuration data of the individual archive tags.

Implementation Concept

A separate layout is configured in the *Report Designer* editor. This layout is not based on any existing layout.

Implementation in the Report Designer

Step	Procedure: Implementation in the Report Designer
1	<p>Creation of a new layout via a  on the <i>Page Layout</i> entry in the <i>WinCC Explorer</i>.</p> <p>In the right window, this new layout, with the default name <i>NewRPL00.RPL</i>, will be added to the existing page layouts. Via a  on this name, rename it to <i>tlg_cs.rpl</i>.</p>
2	<p>Open the new layout in the <i>Report Designer</i>.</p> <p>In the properties dialog of the layout, which is accessed via a  on the blank space of the report page, at the <i>Miscellaneous</i> property, a <i>Cover Sheet</i> is specified. The remaining settings are kept.</p>
3	<p>In the static part of the report, various <i>Static Objects</i> and <i>System Objects</i> are configured for the header and the footer.</p> <p>The design of the cover sheet is only a suggestion for your own design.</p>

Step	Procedure: Implementation in the Report Designer
4	<p>In the dynamic part of the report, a <i>Dynamic Object</i> → <i>Dynamic Table</i> is configured. In the sample, this is the <i>DynTable1</i> object.</p> <p>After the object has been placed in the report, the <i>Connect</i> dialog will be displayed. From the <i>Tag Logging CS</i> folder, select the <i>Process Value Archive</i> entry. Close the dialog box by clicking on <i>OK</i>.</p> 
5	<p>In the <i>Connect</i> tab of the table's properties dialog, several selection options are available.</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Archive names <input checked="" type="checkbox"/> Process archive data
6	<p>Via a  on one of these entries, the corresponding dialog for the data selection is accessed. If a selection is made, it is symbolized by a red check-mark.</p> <p>In the dialog for the selection of the desired archives, the check-box <i>All Archives</i> is selected. In the dialog for the selection of the desired archive data, all listed data selections are check-marked.</p> <p>The settings made in the <i>Report Designer</i> are saved.</p>

Note:

The procedure for creating a new print job and for its activation from the *WinCC Explorer* and runtime is detailed in the first sample of the *Report Designer* chapter at *Creation of a Print Job*. The settings can be made in the same manner.

Note for the General Application

The following adaptations must be made before the general application:

- The layout created can be used without changes.

4.3.4 Printing Out Trend Windows in Runtime (ex_3_chapter_01a.pdl)

Task Definition



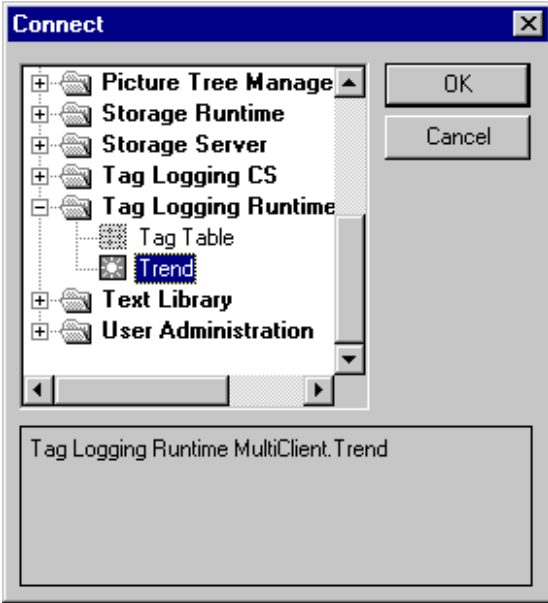
A trend window is to be printed out in runtime. A time range for the data to be printed can be set.








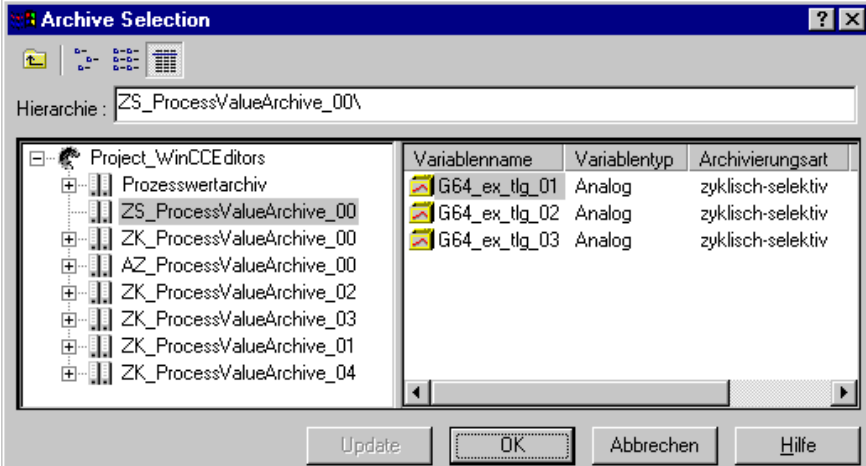
This sample is based on the sample Cyclic-Selective Archiving (ex_3_chapter_01a.pdl) in the *Tag Logging* chapter. It is used to print out the table displayed in this sample.

Implementation Concept






A separate layout is configured in the *Report Designer* editor. The time selection is not made in the layout, but in runtime via a project function. This function will perform the time selection directly in the print job.

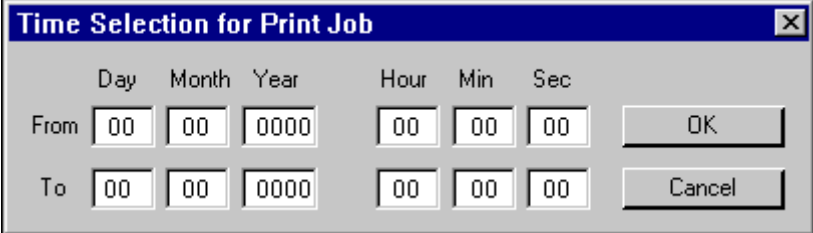

Implementation in the Report Designer

Step	Procedure: Implementation in the Report Designer
1	<p>Creation of a new layout via a  on the <i>Page Layout</i> entry in the <i>WinCC Explorer</i>.</p> <p>In the right window, a new layout is added to the existing ones. The default name of this layout is changed via a  to <i>tlg_ZS_PA_00.rpl</i>.</p>
2	<p>Open the new layout in the <i>Report Designer</i>.</p> <p>In the static part of the report, various <i>Static Objects</i> and <i>System Objects</i> are configured for the header and the footer.</p>
3	<p>In the dynamic part of the report, three <i>Dynamic Objects</i> → <i>Dynamic Metafiles</i> are configured. In this sample, these are the objects <i>DynMetafile1</i>, <i>DynMetafile2</i> and <i>DynMetafile3</i>.</p> <p>After an object has been placed in the report, the <i>Connect</i> dialog will be displayed. For all three objects, the <i>Tag Trend</i> entry is selected from the <i>Tag Logging Runtime</i> folder. The dialogs are closed by clicking on <i>OK</i>.</p> 

Step	Procedure: Implementation in the Report Designer
4	<p>In the <i>Connect</i> tab of the <i>Dynamic Metafile's</i> properties dialog, several selection options are available.</p> <ul style="list-style-type: none">  Time range  Tag Selection  Format <p>Via a  on one of these entries, the corresponding dialog for the data selection is accessed. If a selection is made, it is symbolized by a red check-mark. A <i>Time Selection</i> is not made.</p> <p>In the dialog pertaining to the <i>Tag Selection</i>, the <i>Tag Selection for Reporting</i> dialog is opened via  → <i>Edit</i>.</p> 
5	<p>The settings made in the <i>Report Designer</i> are saved. Via  → <i>Add</i>, the <i>Archive Data Selection</i> dialog is opened. The archive <i>ZS_ProcessValueArchive_00</i> and the tag <i>G64_ex_tlg_01</i> are selected. The dialogs can be closed with <i>OK</i>. In the same way, configure the remaining two tags for the other objects.</p> 

Creation of a Print Job

Step	Procedure: Creation of a Print Job
1	<p>In the <i>WinCC Explorer</i>, a new print job is created via a  on the <i>Print Job</i> entry.</p>  <p>In the right window, this new print job, with the default name <i>Print Job001</i>, will be added to the existing print jobs. Via a  or a  on this print job, its properties dialog is opened.</p> <p>In the <i>Print Job</i> tab, the <i>Name</i> <i>Printjob_ZS_PA_00</i> is entered. As the <i>Layout</i>, the previously created <i>tlg_ZS_PA_00.rpl</i> layout is specified.</p> <p>In the <i>Selection</i> tab, the print range is set. In the <i>Page Range</i> field, the <i>All</i> radio-button is selected. In the <i>Time Range</i> field, the <i>Absolute</i> radio-button is selected. A specific time range is not set, this will be done later in runtime.</p> <p>In the <i>Printer Setup</i> tab, the printer to be used is specified.</p>
2	<p>In the sample project, a preview of the print job can be activated via a <i>Windows Object</i> → <i>Button</i>. This is the <i>Button13</i> object in the <i>ex_3_chapter_01a.pdl</i> picture.</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, create a <i>C-Action</i> that starts a preview of the corresponding print job.</p> 

Step	Procedure: Creation of a Print Job
3	<p>To make a time selection, a settings dialog is needed. This dialog is configured as a separate picture - in the sample, this is the <i>ex_5_window_02.PDL</i> picture.</p> <p>In this picture, six <i>Smart Objects</i> → <i>I/O Fields</i> each are configured for the entry of the start and end time. These are the I/O Field1 to I/O Field6 objects for the start time entry and the I/O Field7 to I/O Field12 objects for the end time entry. To buffer the entries made, a tag of the <i>Unsigned 16-Bit Value</i> type is configured in Tag Management for each <i>I/O Field</i>. In this sample, these are the tags <i>U16i_ex_rep_f1</i> to <i>U16i_ex_rep_f6</i> for the start time and the tags <i>U16i_ex_rep_t1</i> to <i>U16i_ex_rep_t6</i> for the end time. For each <i>I/O Field</i>, a <i>Tag Connection</i> is created at <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i> to its corresponding tag.</p> <p>At <i>Properties</i> → <i>Output/Input</i> → <i>Output Format</i>, set the format to <i>099</i> for all <i>I/O Fields</i>, except for the <i>I/O Fields</i> containing the year. These fields use the output format <i>09999</i>.</p> 
4	<p>At <i>Events</i> → <i>Miscellaneous</i> → <i>Open Picture</i> of the <i>ex_5_window_02.PDL</i> picture, a <i>C-Action</i> is created that supplies the tags for setting the time with predefined time values. As the end time, the current system time is set and as the start time, the current system time minus one minute.</p>
5	<p>In the <i>ex_5_window_02.PDL</i> picture, two <i>Windows Objects</i> → <i>Buttons</i> are configured. In the sample, these are the <i>Button1</i> and <i>Button2</i> objects.</p> <p><i>Button2</i> is the Cancel button. At <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, a <i>Direct Connection</i> is created, which switches the <i>Constant 0</i> to the <i>Display</i> of the <i>Current Window</i>. <i>Button1</i> is the OK button. It also receives a <i>Direct Connection</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> for closing the window. At <i>Events</i> → <i>Mouse</i> → <i>Press left</i>, a <i>C-Action</i> is configured. This action calls a previously created project function for making a time selection in a certain print job. The name of the print job is read from a tag of the <i>Text Tag 16-Bit Character Set</i> type that has to be created in <i>Tag Management</i>. In the sample, this is the <i>T16i_ex_rep_00</i> tag.</p>
6	<p>To display the created picture in the <i>ex_3_chapter_01a.PDL</i> picture, a <i>Smart Object</i> → <i>Picture Window</i> must be created. In the sample, this is the <i>Picture Window1</i> object. At <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>, set the previously created picture <i>ex_5_window_02.PDL</i>. Set the <i>Display</i> property to <i>No</i>.</p>
7	<p>To make the <i>Picture Window</i> visible, an additional <i>Windows Object</i> → <i>Button</i> is required - in the sample, this is the <i>Button12</i> object. This object receives a <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>, which writes the name of the print job to be processed to the <i>T16i_ex_rep_00</i> tag and makes the <i>Picture Window1</i> object visible.</p> 

C-Action at the OK Button

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    ModifyPrintJob(TimeFrom(),
                  TimeTo(),
                  GetTagChar("T16i_ex_rep_00"));
}
```

- Call of the project function *ModifyPrintJob*. This function requires as the transfer parameters two time values in the form of the SYSTEMTIME structure. These values are determined by two project functions from the internal tags storing the time values (*TimeFrom()* and *TimeTo()*). Additionally, the name of the print job processed is required. This name is stored in the *T16i_ex_rep_00* tag.

Project Function ModifyPrintJob

```

BOOL ModifyPrintJob(SYSTEMTIME st1,SYSTEMTIME st2,char jobname[200])
{
    BOOL          fRet;
    PCMN_ERROR    pError;
    HPROPERTIES   hProp;
    LPVOID        ptr1,ptr2;
    DWORD        typ;
    DWORD        dwVal;
    char          propname1[200],propname2[200];
    TCHAR        g_szProj[MAX_PATH+1];

    typ = VT_DATE;
    strcpy( propname1, "ABSOLUTESELECTIONFROM");
    strcpy( propname2, "ABSOLUTESELECTIONTO");
    ptr1 = (LPVOID)&st1;
    ptr2 = (LPVOID)&st2;

    //-----get project path
    if( !DMGetRuntimeProject( g_szProj, MAX_PATH, pError)) {
        printf("Error DMGetRuntimeProject(...)\r\n");
        return FALSE;
    }

    //-----create property handle
    hProp = RPJCreatePropertyHandle ( g_szProj, pError );
    if( !hProp) {
        printf("Error RPJCreatePropertyHandle(...)\r\n");
        return FALSE;
    }

    //-----get job properties
    if ( !RPJGetJobProps ( hProp, jobname, pError )) {
        printf("Error RPJGetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
    }

    //-----set property
    if ( !RPJSetProperty ( hProp, propname1, ptr1,
        (VARTYPE) typ, 200, pError )) {
        printf("Error RPJSetProperty(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
    }

    //-----save job properties
    if ( !RPJSetJobProps ( hProp, jobname, pError)) {
        printf("Error RPJSetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
    }

    //-----get job properties
    if ( !RPJGetJobProps ( hProp, jobname, pError )) {
        printf("Error RPJGetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
    }

    //-----set property
    if ( !RPJSetProperty ( hProp, propname2, ptr2,
        (VARTYPE) typ, 200, pError )) {
        printf("Error RPJSetProperty(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
    }

    //-----save job properties
    if ( !RPJSetJobProps ( hProp, jobname, pError)) {
        printf("Error RPJSetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
    }

    //-----delete property handle
    fRet = RPJDeletePropertyHandle ( hProp, pError);
    return TRUE;
}

```

- As the *st1* and *st2* transfer parameters, this function receives the start and end times to be set in the form of a SYSTEMTIME structure.
- Determination of the project path using the *DMGetRuntimeProject* function.
- Setting and saving of the start time. This is the property *ABSOLUESELECTIONFROM*.
- Setting and saving of the end time. This is the property *ABSOLUESELECTIONTO*.

Note for the General Application

The following adaptations must be made before the general application:

- In the layout used for printing the trend profiles, the archive to be reported as well as the archive tags to be printed must be adapted.
- The dialog for the time selection can be applied without changes. For the operation, the project functions *ModifyPrintJob*, *TimeFrom* and *TimeTo* are required. The tags for buffering the time values must be created using the same names. Otherwise, the project functions *TimeFrom* and *TimeTo* must be adapted. In order to use the dialog for several print jobs, the creation of a text tag for storing the print job name is recommended.

4.3.5 Printing Out Tables in Runtime (ex_3_chapter_01c.pdl)

Task Definition



A table is to be printed out in runtime. A time range for the data to be printed can be set. This sample is based on the sample User-Defined Table Layout (ex_3_chapter_01c.pdl) in the *Tag Logging* chapter. It is used to print out the table displayed in this sample.

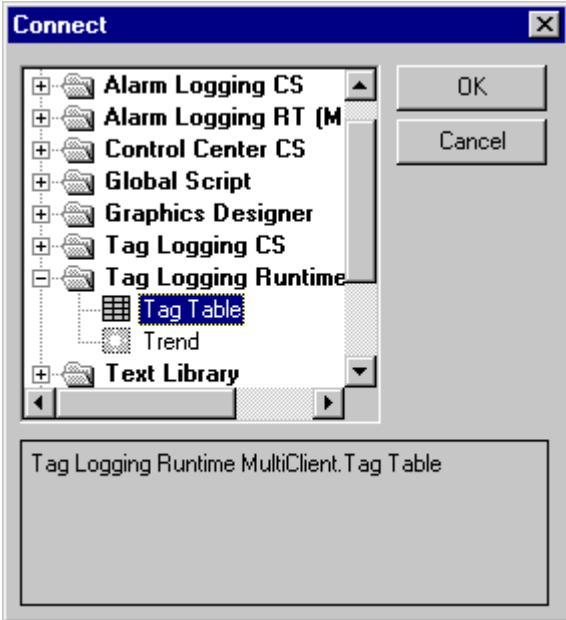
Implementation Concept





A new page layout is configured in the *Report Designer* editor. The time selection is not made in the layout, but in runtime via a project function. This function will perform the time selection directly in the print job.

The procedure for making a time selection in runtime is detailed in the previous sample at *Creation of a Print Job*.


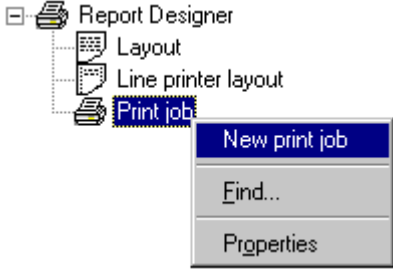




Implementation in the Report Designer

Step	Procedure: Implementation in the Report Designer
1	Creation of a new layout via a  on the <i>Page Layout</i> entry in the <i>WinCC Explorer</i> . In the right window, a new layout is added to the existing ones. The default name of this layout is changed via a  to <i>tlg_ZK_PA_02.rpl</i> .
2	Open the new layout in the <i>Report Designer</i> . In the static part of the report, various <i>Static Objects</i> and <i>System Objects</i> are configured for the header and the footer.

Step	Procedure: Implementation in the Report Designer
3	<p>In the dynamic part of the report, a <i>Dynamic Object</i> → <i>Dynamic Table</i> is configured. In the sample, this is the <i>DynTable1</i> object.</p> <p>After the object has been placed in the report, the <i>Connect</i> dialog will be displayed. From the <i>Tag Logging Runtime</i> folder, the <i>Tag Table</i> entry is selected. Close the dialog box by clicking on <i>OK</i>.</p>  <p>The screenshot shows a dialog box titled "Connect" with a close button (X) in the top right corner. The main area contains a tree view of project components. The components listed are: Alarm Logging CS, Alarm Logging RT (M), Control Center CS, Global Script, Graphics Designer, Tag Logging CS, Tag Logging Runtime, Tag Table (selected), Trend, and Text Library. To the right of the tree view are two buttons: "OK" and "Cancel". Below the tree view, there is a text field containing the path "Tag Logging Runtime MultiClient.T.ag T.able".</p>

Step	Procedure: Implementation in the Report Designer				
4	<p>In the <i>Connect</i> tab of the <i>Dynamic Metafile</i>'s properties dialog, several selection options are available.</p> <ul style="list-style-type: none">  Time range  Tag Selection <p>Via a  on one of these entries, the corresponding dialog for the data selection is accessed. If a selection is made, it is symbolized by a red check-mark. A <i>Time Range</i> is not specified.</p> <p>Via a  → <i>Edit</i> → <i>Add</i>, the dialog for the archive selection is archived. In this dialog, the <i>ZK_ProcessValueArchive_00</i> archive and its archive tags are selected. The dialogs can be closed with OK.</p> <div data-bbox="483 674 1349 1184" style="border: 1px solid black; padding: 5px;"> <p>Tag Logging Runtime: Tag selection for reporting ✕</p> <p>Current selection and sequence:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Variable</th> </tr> </thead> <tbody> <tr> <td>ZK_ProcessValueArchive_02\MaximumValue</td> </tr> <tr> <td>ZK_ProcessValueArchive_02\MeanValue</td> </tr> <tr> <td>ZK_ProcessValueArchive_02\MinimumValue</td> </tr> </tbody> </table> <div style="float: right; text-align: right;"> <input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Add..."/> <input type="button" value="Move Up"/> <input type="button" value="Move Down"/> <input type="button" value="Delete"/> <input type="button" value="Properties..."/> </div> <p style="font-size: small; text-align: center;">This dialog box allows you to select tags for reporting from existing Tag Logging archives.</p> </div> <p>The settings made in the <i>Report Designer</i> are saved.</p>	Variable	ZK_ProcessValueArchive_02\MaximumValue	ZK_ProcessValueArchive_02\MeanValue	ZK_ProcessValueArchive_02\MinimumValue
Variable					
ZK_ProcessValueArchive_02\MaximumValue					
ZK_ProcessValueArchive_02\MeanValue					
ZK_ProcessValueArchive_02\MinimumValue					

Creation of a Print Job

Step	Procedure: Creation of a Print Job
1	<p>In the <i>WinCC Explorer</i>, a new print job is created via a .</p>  <p>In the right window, this new print job, with the default name <i>Print Job001</i>, will be added to the existing print jobs. Via a  or a  on this print job, its properties dialog is opened.</p> <p>In the <i>Print Job</i> tab, the Name <i>Printjob_ZK_PA_02</i> is entered. As the <i>Layout</i>, the previously created <i>tlg_ZK_PA_02.rpl</i> layout is specified.</p> <p>In the <i>Selection</i> tab, the print range is set. In the <i>Page Range</i> field, the <i>All</i> radio-button is selected. In the <i>Time Range</i> field, the <i>Absolute</i> radio-button is selected. A specific time range is not set, this will be done later in runtime.</p> <p>In the <i>Printer Setup</i> tab, the printer to be used is specified.</p>
2	<p>The procedure for making a time selection in runtime is detailed in the previous sample at <i>Creation of a Print Job</i>.</p> <p>Adaptations must be made for the button displayed below in the <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i>.</p>  <p>The text tag <i>T16i_ex_rep_00</i> must be supplied with the name of the just created print job.</p> <p>For the button starting the print preview, the print job name in the function call of the <i>C-Action</i> at <i>Events</i> → <i>Mouse</i> → <i>Mouse Action</i> must also be changed.</p> 

Note for the General Application

The following adaptations must be made before the general application:

- The layout designed can be used directly after adapting the archive selection.

4.3.6 Message Sequence Report (ex_3_chapter_02b.pdl)

Task Definition


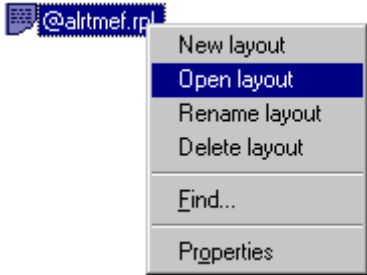
A message sequence report is to be created. After a layout page has been filled completely, the message sequence report is to be printed automatically.

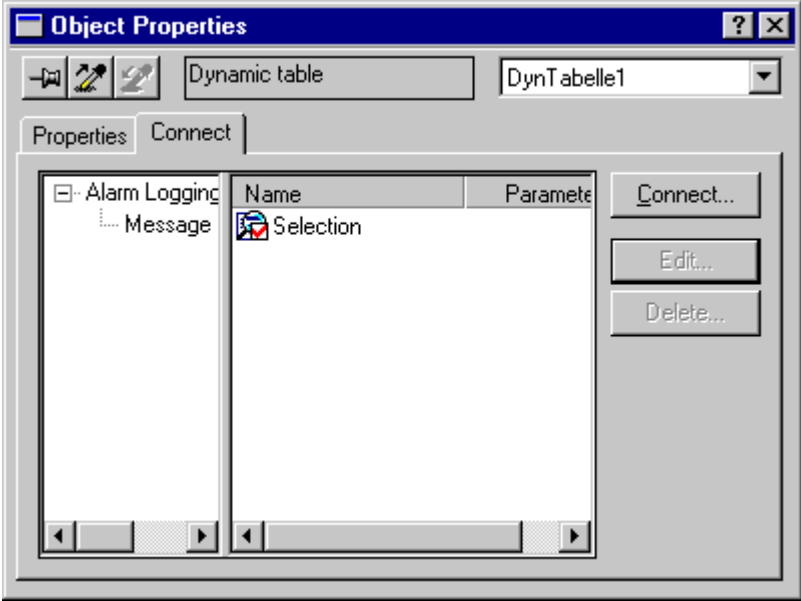



This sample is based on the sample Message Window (ex_3_chapter_02b.pdl) in the *Alarm Logging* chapter. In that sample, a toolbar button has already been included for the report functions in the message window used and the message sequence report been activated.

Implementation Concept





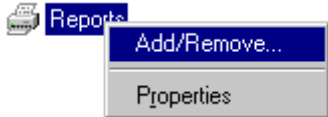

A system layout as well as a system print job matching the requirements specified are available. This is the *@alrtmef.rpl* layout and the *@Report Alarm Logging RT Message sequence* print job. The layout is copied and adapted to your needs. As the print job, the system print job must be used - only the layout used by this system print job is changed.

Implementation in the Report Designer

Step	Procedure: Implementation in the Report Designer
1	<p>Open the system layout <i>@alrtmef.rpl</i> in the <i>WinCC Explorer</i> via a  on its name.</p>  <p>Via the <i>File</i> → <i>Save As...</i> menus, the system layout is saved under a different name. In this sample, the name <i>alg_mef.rpl</i> is used.</p>
2	<p>The static part of the report contains a header and a footer. The elements of the static part can be adapted to meet your requirements.</p>

Step	Procedure: Implementation in the Report Designer
3	<p>The dynamic part of the report consists of a <i>Dynamic Object</i> → <i>Dynamic Table</i>. In the sample, this is the <i>DynTable1</i> object.</p> <p>The properties dialog of the <i>DynTable1</i> object is opened and the <i>Connect</i> tab selected. The table is already connected to the <i>Message Sequence Report</i> of <i>Alarm Logging Runtime</i>. A selection has also been already performed.</p> 
4	<p>Via  → <i>Edit</i> or a  on the <i>Selection</i> entry, the dialog for the message block selection is opened. In this dialog, the system blocks <i>Date</i>, <i>Time</i>, <i>Number</i> and <i>Loop in Alarm</i> are already selected. In this sample, all remaining message blocks are selected via the button displayed below.</p> <p>The dialog is closed by clicking on <i>OK</i>. The layout is saved.</p> 

Adaptation of the Print Job

Step	Procedure: Adaptation of the Print Job
1	<p>Open the system print job <i>@Report Alarm Logging RT Message sequence</i> via a  or a  on its name in the <i>WinCC Explorer</i>.</p> 
2	<p>In the <i>Print Job</i> tab, the just created <i>Layout alrtmef.rpl</i> is specified. In addition, the desired printer is set in the <i>Printer Setup</i> tab. No other adaptations are needed. Close the dialog box by clicking on <i>OK</i>.</p>
3	<p>The message sequence report must be activated in the <i>Alarm Logging</i> editor. To do this, open this editor. Via a  on the <i>Reports</i> entry, the <i>Assigning Report Parameters</i> dialog is opened.</p>  <p>In this dialog, the <i>Check-Box</i> for the message sequence report is selected.</p>
4	<p>If the user does not activate the print job manually, the message sequence report will be printed out automatically as soon as a layout page is full.</p> <p>To allow the user to start the message sequence report at any time, a toolbar button must be included while configuring the message window template. This is the button function named <i>Report Functions</i>. In the sample project, this function has already been selected in the <i>MessageWindow_04</i> template used in the <i>ex_3_chapter_02b.pdl</i> picture.</p>  <p>The settings made in <i>Alarm Logging</i> are saved.</p>

Note for the General Application

The following adaptations must be made before the general application:

- In the layout created, the message blocks desired for the message sequence report must be adapted.

4.3.7 Message Sequence Report on a Line Printer





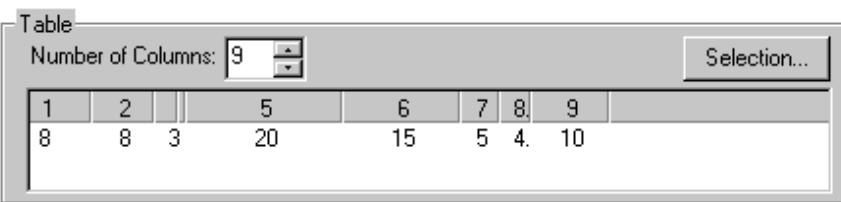
Task Definition

A message sequence report is to be designed that is suitable for the output on a line printer. If a message to be reported arrives, it is to be printed automatically.



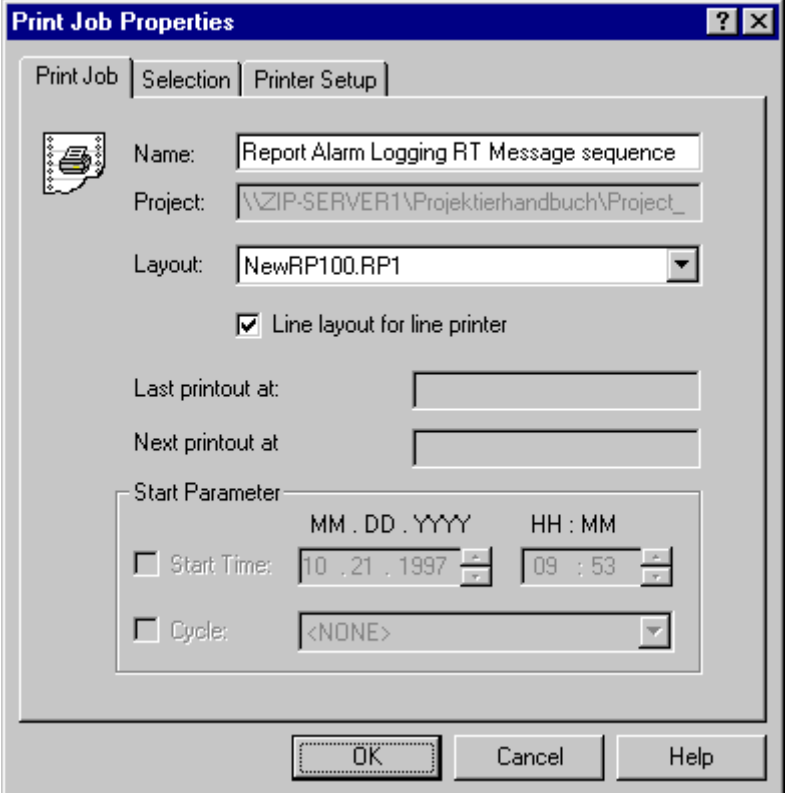

Implementation Concept

A line layout is created. This layout is then specified for the system print job *@Report Alarm Logging RT Message sequence*.

Creation of a Line Layout

Step	Procedure: Creation of a Line Layout
1	<p>Creation of a new line layout via a  on the appropriate entry in the <i>WinCC Explorer</i>.</p> 
2	<p>A new line layout with the default name <i>NewRP100.RPI</i> is created. This name is kept for the sample project. With a  on the name of the new line layout in the right window, the <i>Line Layout Editor</i> is opened.</p> <p>In this editor, general settings pertaining to page margins, headers, footers, etc. can be made.</p> <p>In the <i>Table</i> field, a dialog for the selection of the desired message blocks in the message sequence report is opened via a  → <i>Selection</i>. In this sample, all available message blocks are selected.</p>
3	<p>The number of columns as well as their width is matched automatically to the selected message blocks and their order.</p>  <p>The settings made are saved and the line layout editor closed.</p>

Adaptation of the Print Job

Step	Procedure: Adaptation of the Print Job
1	Open the system print job <i>@Report Alarm Logging RT Message sequence</i> via a  or a  on its name in the <i>WinCC Explorer</i> .
2	<p>In the <i>Print Job</i> tab, the <i>Line Layout for Line Printer</i> check-box is selected and the just created <i>Layout NewRP100.RP1</i> specified. In addition, the desired line printer is set in the <i>Printer Setup</i> tab. No other adaptations are needed. Close the dialog box by clicking on <i>OK</i>.</p> 
3	The message sequence report must be activated in the <i>Alarm Logging</i> editor. To do this, open this editor. Via a  on the <i>Reports</i> entry, the <i>Assigning Report Parameters</i> dialog is opened. In this dialog, the <i>Check-Box</i> for the message sequence report is selected.

Note for the General Application

The following adaptations must be made before the general application:

- The desired page settings and message blocks to be printed must be adapted in the line layout editor.

4.3.8 Message Archive Report (ex_3_chapter_02c.pdl)

Task Definition




A message archive report is to be created. The print job is to be activated by the user via a button.

This sample is based on the sample Message Archiving (ex_3_chapter_02c.pdl) in the *Alarm Logging* chapter. In that sample, a toolbar button has already been included for the report functions of the short-term (revolving) archive window and the message sequence report been activated.






Implementation Concept

A system layout as well as a system print job matching the requirements specified are available. This is the *@alrtuma.rpl* layout and the *@Report Alarm Logging RT Revolving archive* print job. The layout is copied and adapted to your needs. As the print job, the system print job must be used - only the layout used by this system print job is changed.

Implementation in the Report Designer

Step	Procedure: Implementation in the Report Designer
1	<p>Open the system layout <i>@alrtuma.rpl</i> in the <i>WinCC Explorer</i> via a  on its name in the right window.</p> <p>Via the <i>File</i> → <i>Save As...</i> menus, the system layout is saved under a different name. In this sample, the name <i>alg_uma</i> is used.</p>
2	<p>The static part of the report contains a header and a footer.</p> <p>The elements of the static part can be adapted to meet your requirements.</p>
3	<p>The dynamic part of the report consists of a <i>Dynamic Object</i> → <i>Dynamic Table</i>. In the sample, this is the <i>DynTable1</i> object.</p> <p>The properties dialog of the <i>DynTable1</i> object is opened and the <i>Connect</i> tab selected. The table is already connected to the <i>Short-Term Archive Report</i> of <i>Alarm Logging Runtime</i>. A selection has also been already performed.</p>
4	<p>Via the  → <i>Edit</i> button or a  on the <i>Selection</i> entry, the dialog for the message block selection is opened. In this dialog, the system blocks <i>Date</i>, <i>Time</i> and <i>Number</i> are already selected. In this sample, all remaining message blocks are selected via the button displayed below.</p> <p>The dialog is closed by clicking on <i>OK</i>. The layout is saved.</p> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-top: 10px;">->></div>

Adaptation of the Print Job

Step	Procedure: Adaptation of the Print Job
1	Open the system print job <i>@Report Alarm Logging RT Revolving archive</i> via a  or a  on its name in the <i>WinCC Explorer</i> .
2	In the <i>Print Job</i> tab, the just created <i>Layout alrtuma.rpl</i> is specified. In addition, the desired printer is set in the <i>Printer Setup</i> tab. No other adaptations are needed. Close the dialog box by clicking on <i>OK</i> .
3	<p>The archive report must be activated in the <i>Alarm Logging</i> editor. To do this, open this editor. Via a  on the <i>Reports</i> entry, the <i>Assigning Report Parameters</i> dialog is opened.</p>  <p>In this dialog, the <i>Check-Box</i> for the archive report is selected. The settings made in <i>Alarm Logging</i> are saved.</p>
4	<p>To allow the user to start the message sequence report at any time, a toolbar button must be included while configuring the message window template. If a user-defined toolbar is used, the pressing of this button must be simulated using the corresponding standard function. This is the standard function <i>ACX_OnBtnPrint()</i>. In the sample project, this button is already configured in the <i>ex_3_chapter_02c.pdl</i> picture.</p> 


Note for the General Application

The following adaptations must be made before the general application:

- In the layout created, the message blocks desired for the archive report must be adapted.

4.4 OLE Communication with EXCEL



In runtime, the samples pertaining to this topic are accessed by selecting the button displayed above using the . These samples are configured in the *ex_3_chapter_04.pdl* picture and the *OLE_Communication.xls* Excel folder.

4.4.1 Reading and Writing Tag Values (ex_3_chapter_04.pdl)

Task Definition

The values of internal tags of various types are to be written to an Excel spreadsheet. In the second column of the spreadsheet, the setpoint values for these tags is to be entered. These values are then written back to the WinCC project.

Implementation Concept




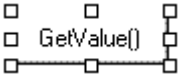
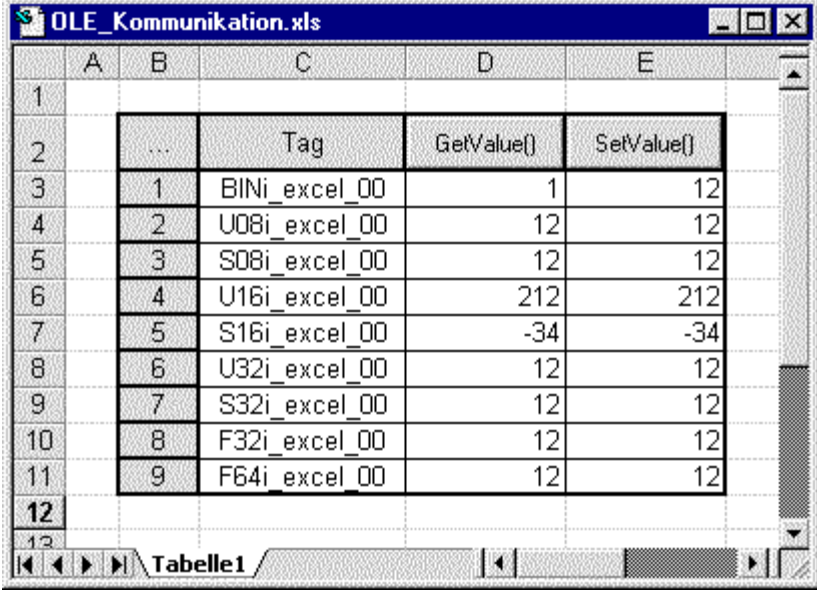
In a picture, one I/O field is configured for each tag, which displays the value of the tag and can be assigned a value.

In Excel (Version 8.0), a spreadsheet is created. In a column of this table, the names of the tags to be read and written to are entered. Two buttons are added to the spreadsheet. Macros are assigned to these buttons, which read in the tag names to be processed and either read the corresponding tag values or assign setpoint values to them.

Implementation in the WinCC Project

Step	Procedure: Implementation in the WinCC Project																				
1	<p>In Tag Management, several tags of various types are created. In the sample, the following tags were used:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>BINi_excel_00</td> <td>Binary Tag</td> </tr> <tr> <td>F32i_excel_00</td> <td>Floating-point number 32-bit IEEE 754</td> </tr> <tr> <td>F64i_excel_00</td> <td>Floating-point number 64-bit IEEE 754</td> </tr> <tr> <td>S16i_excel_00</td> <td>Signed 16-bit value</td> </tr> <tr> <td>S32i_excel_00</td> <td>Signed 32-bit value</td> </tr> <tr> <td>S08i_excel_00</td> <td>Signed 8-bit value</td> </tr> <tr> <td>U16i_excel_00</td> <td>Unsigned 16-bit value</td> </tr> <tr> <td>U32i_excel_00</td> <td>Unsigned 32-bit value</td> </tr> <tr> <td>U08i_excel_00</td> <td>Unsigned 8-bit value</td> </tr> </tbody> </table>	Name	Type	BINi_excel_00	Binary Tag	F32i_excel_00	Floating-point number 32-bit IEEE 754	F64i_excel_00	Floating-point number 64-bit IEEE 754	S16i_excel_00	Signed 16-bit value	S32i_excel_00	Signed 32-bit value	S08i_excel_00	Signed 8-bit value	U16i_excel_00	Unsigned 16-bit value	U32i_excel_00	Unsigned 32-bit value	U08i_excel_00	Unsigned 8-bit value
Name	Type																				
BINi_excel_00	Binary Tag																				
F32i_excel_00	Floating-point number 32-bit IEEE 754																				
F64i_excel_00	Floating-point number 64-bit IEEE 754																				
S16i_excel_00	Signed 16-bit value																				
S32i_excel_00	Signed 32-bit value																				
S08i_excel_00	Signed 8-bit value																				
U16i_excel_00	Unsigned 16-bit value																				
U32i_excel_00	Unsigned 32-bit value																				
U08i_excel_00	Unsigned 8-bit value																				
2	<p>In a picture, a <i>Smart Object</i> → <i>I/O Field</i> is created for each tag. At <i>Properties</i> → <i>Output/Input</i> → <i>Output Value</i>, a <i>Tag Connection</i> is created to each tag.</p>																				

Implementation in Excel (Version 8.0)

Step	Procedure: Implementation in Excel																																																																														
1	<p>Create a new Excel folder. In this sample, the folder is named <i>OLE_Communication.xls</i>.</p> <p>In a spreadsheet, fill out a column with the names of the tags to be processed.</p>																																																																														
2	<p>Configure a button for reading the tag values.</p> <p>For this purpose, activate the <i>Control Toolbox</i> toolbar via the <i>View</i> → <i>Toolbars</i> menus, if it has not been activated yet. Select the Command Button control element and place it in the table.</p> 																																																																														
3	<p>The properties of this control element can be set in the dialog accessed by the button displayed below. As the object name, the sample uses <i>GetValue</i> and as the caption, <i>GetValue()</i>.</p> 																																																																														
4	<p>With a  on the button created, the Visual Basic Editor is opened and the procedure to be executed can be entered.</p> 																																																																														
5	<p>As described in step 2, configure another button for writing tag values.</p>  <table border="1" data-bbox="527 1159 1339 1747"> <thead> <tr> <th></th> <th>A</th> <th>B</th> <th>C</th> <th>D</th> <th>E</th> </tr> </thead> <tbody> <tr> <td>1</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>2</td> <td></td> <td>...</td> <td>Tag</td> <td>GetValue()</td> <td>SetValue()</td> </tr> <tr> <td>3</td> <td>1</td> <td></td> <td>BINi_excel_00</td> <td>1</td> <td>12</td> </tr> <tr> <td>4</td> <td>2</td> <td></td> <td>U08i_excel_00</td> <td>12</td> <td>12</td> </tr> <tr> <td>5</td> <td>3</td> <td></td> <td>S08i_excel_00</td> <td>12</td> <td>12</td> </tr> <tr> <td>6</td> <td>4</td> <td></td> <td>U16i_excel_00</td> <td>212</td> <td>212</td> </tr> <tr> <td>7</td> <td>5</td> <td></td> <td>S16i_excel_00</td> <td>-34</td> <td>-34</td> </tr> <tr> <td>8</td> <td>6</td> <td></td> <td>U32i_excel_00</td> <td>12</td> <td>12</td> </tr> <tr> <td>9</td> <td>7</td> <td></td> <td>S32i_excel_00</td> <td>12</td> <td>12</td> </tr> <tr> <td>10</td> <td>8</td> <td></td> <td>F32i_excel_00</td> <td>12</td> <td>12</td> </tr> <tr> <td>11</td> <td>9</td> <td></td> <td>F64i_excel_00</td> <td>12</td> <td>12</td> </tr> <tr> <td>12</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		A	B	C	D	E	1						2		...	Tag	GetValue()	SetValue()	3	1		BINi_excel_00	1	12	4	2		U08i_excel_00	12	12	5	3		S08i_excel_00	12	12	6	4		U16i_excel_00	212	212	7	5		S16i_excel_00	-34	-34	8	6		U32i_excel_00	12	12	9	7		S32i_excel_00	12	12	10	8		F32i_excel_00	12	12	11	9		F64i_excel_00	12	12	12					
	A	B	C	D	E																																																																										
1																																																																															
2		...	Tag	GetValue()	SetValue()																																																																										
3	1		BINi_excel_00	1	12																																																																										
4	2		U08i_excel_00	12	12																																																																										
5	3		S08i_excel_00	12	12																																																																										
6	4		U16i_excel_00	212	212																																																																										
7	5		S16i_excel_00	-34	-34																																																																										
8	6		U32i_excel_00	12	12																																																																										
9	7		S32i_excel_00	12	12																																																																										
10	8		F32i_excel_00	12	12																																																																										
11	9		F64i_excel_00	12	12																																																																										
12																																																																															

Procedure for Reading Tag Values

```

Rem Read Tag Values in WinCC-Project
Private Sub GetValue_Click()
    Dim mcp As Object
    Dim var As String
    Dim value As Variant
    Dim cell As Variant
    Dim i As Integer

    Set mcp = CreateObject("WinCC-Runtime-Project")

    Cell = "C3"
    i = 1

    Do While Not Range(cell) = ""
        var = Range(cell)
        value = mcp.GetValue(var)
        Range("D" & 2 + i).value = value
        cell = "C" & 3 + i
        i = i + 1
    Loop
End Sub

```

- Generation of a WinCC object, which is stored in the *mcp* tag.
- In a loop, the individual cells of the column containing the names of the tags to be processed are read. In the WinCC project, the tags are read with the *GetValue()* function and their values written to the column next to it. The loop is repeated until the first blank cell is reached.

Procedure for Writing Tag Values

```

Rem Set Tag Values in WinCC-Project
Private Sub SetValue_Click()
    Dim mcp As Object
    Dim var As String
    Dim value As Variant
    Dim cell As Variant
    Dim i As Integer
    Dim bRet As Integer

    Set mcp = CreateObject("WinCC-Runtime-Project")

    Cell = "C3"
    i = 1

    Do While Not Range(cell) = ""
        var = Range(cell)
        value = Range("E" & 2 + i).value
        bRet = mcp.SetValue(var, value)
        cell = "C" & 3 + i
        i = i + 1
    Loop
End Sub

```

- Generation of a WinCC object, which is stored in the *mcp* tag.
- In a loop, the individual cells of the column containing the names of the tags to be processed are read. In addition, a column containing the tag values to be set is read. The tags are written to the WinCC project with the *SetValue()* function. The loop is repeated until the first blank cell is reached.

Note for the General Application

The following adaptations must be made before the general application:

- Data is exchanged between WinCC and Excel with the GetValue() and SetValue() functions. The data can be processed in Excel as desired.

4.5 Additional Configurations in the Samples

This chapter describes the additional elements used in some pictures. Their description in the associated samples would be too exhaustive, since they are not directly related to the topic at hand. This chapter completes the description of the sample project.

4.5.1 Picture Index

Task Definition

The order of the last 10 selected pictures in the project is to be stored. Via a back button, the pictures are to be selectable in reverse order. Via a next button, you can move forward in the order again up to the current picture.

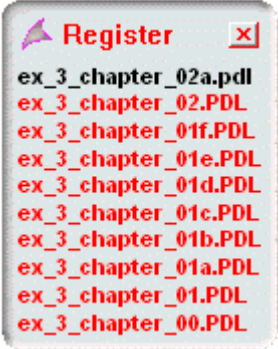
In a separate picture, all pictures of the index are to be displayed in the correct order. From that picture it is also possible to directly select pictures.

Implementation Concept

The picture order is stored in 10 static C variables of a project function. This project function is called every time a picture change is performed. It controls the picture selection via the next and back buttons as well as the direct picture selection.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	<p>Two tags of the <i>Binary Tag</i> type are used. These are the <i>BINi_ex_org_00</i> and <i>BINi_ex_org_01</i> tags. These tags make the next and back buttons operational.</p> <p>Additionally, a tag of the <i>Unsigned 16-Bit Value</i> type is used. In the sample, this is the <i>U16i_ex_org_00</i> tag. This tag stores the current position in the picture index.</p> <p>Also, a tag of the <i>Text Tag 16-Bit Character Set</i> type is used. In the sample, this is the <i>T16x_ex_org_00</i> tag. This tag records the current picture name.</p>
2	<p>A project function for controlling the picture index is available. This is the <i>CreatePictureSequence</i> function. This function is called every time a picture change is performed. This is achieved by a <i>C-Action</i> at <i>Events</i> → <i>Property Topics</i> → <i>Miscellaneous</i> → <i>Picture Name</i> of the <i>workspace</i> object in the <i>ex_0_startpicture_00.pdl</i> picture. Every time this function is called, the new picture name is stored in the index and the existing names are moved back by one position.</p>
3	<p>In the <i>ex_2_keyboard_00.pdl</i> keyboard picture, two control elements for moving backwards and forwards are configured.</p> <p>If one of these objects is pressed, the project function <i>CreatePictureSequence</i> is called as well. This function will then perform the picture change. Via two <i>Smart Objects</i> → <i>Graphic Objects</i>, these two control elements can be made inoperational.</p> <div data-bbox="532 1581 808 1661" data-label="Image"> </div>
4	<p>Via the following button, a picture is called that displays the current picture index. In the sample, this is the <i>ex_9_register_00.PDL</i> picture.</p> <div data-bbox="532 1755 808 1835" data-label="Image"> </div>
5	<p>In this picture, 10 <i>Standard Objects</i> → <i>Static Texts</i> are configured. The</p>

Step	Procedure: Implementation in the Graphics Designer
	<p>default texts are deleted from all objects. At the picture object's <i>Properties</i> → <i>Geometry</i> → <i>Picture Width</i>, a <i>C-Action</i> is created that calls the project function <i>CreatePictureSequence</i>. This action provides the <i>Static Texts</i> with the stored picture names. This <i>C-Action</i> is triggered upon the change of the tag <i>T16x_ex_org_00</i>. This results in the update of the display upon a picture change.</p> <p>For each <i>Static Text</i>, a <i>C-Action</i> is created at <i>Events</i> → <i>Mouse</i> → <i>Press left</i> that calls the project function and performs the picture change to the displayed picture. The currently selected picture is highlighted. This is accomplished via a <i>Dynamic Dialog</i> at <i>Properties</i> → <i>Colors</i> → <i>Font Color</i> for each <i>Static Text</i>.</p> 
6	<p>Via a parameter, the project function recognizes from which position it was called. For this parameter, various constants are defined in the <i>APDEFAP.H</i> file. This file is located in the <i>Library</i> subfolder of the project folder.</p> <p>The following constants were defined:</p> <ul style="list-style-type: none"> • #define REG_INSERTPICTURE 0 • #define REG_BACK 1 • #define REG_FORWARD 2 • #define REG_DIRECT 3 • #define REG_SHOWREGISTER 4

Project Function for Controlling the Picture Index

```

#include "APDEFAP.H"
#define MAX_REG 10
void CreatePictureSequence(char* PicName,int nFlag,int nPos)
{
static char PictureName[MAX_REG][40] = {"","","","","","","","","",""};
int i;
static int pos = 0;
static int st = 0;
static int biz = 0;

if (nFlag==REG_INSERTPICTURE){
if (st == 0 ){
pos = 0;
if (biz < MAX_REG) biz++;
for ( i=(MAX_REG-1) ; i>0 ; i-- ){
strcpy(PictureName[i],PictureName[i-1]);
}
strcpy(PictureName[0],PicName);
}
else st=0;
}
if (nFlag==REG_BACK){
pos++;
if ( pos > (MAX_REG-1) ) pos=(MAX_REG-1);
st = 1;
SetPictureName("ex_0_startpicture_00.PDL",
"workspace",PictureName[pos]);
}
if (nFlag==REG_FORWARD){
pos--;
if ( pos < 0 ) pos=0;
st = 1;
SetPictureName("ex_0_startpicture_00.PDL",
"workspace",PictureName[pos]);
}
if (nFlag==REG_SHOWREGISTER){
SetText("ex_9_register_00.PDL",
"Static Text1",PictureName[0]);
SetText("ex_9_register_00.PDL",
"Static Text2",PictureName[1]);
SetText("ex_9_register_00.PDL",
"Static Text3",PictureName[2]);
SetText("ex_9_register_00.PDL",
"Static Text4",PictureName[3]);
SetText("ex_9_register_00.PDL",
"Static Text5",PictureName[4]);
SetText("ex_9_register_00.PDL",
"Static Text6",PictureName[5]);
SetText("ex_9_register_00.PDL",
"Static Text7",PictureName[6]);
SetText("ex_9_register_00.PDL",
"Static Text8",PictureName[7]);
SetText("ex_9_register_00.PDL",
"Static Text9",PictureName[8]);
SetText("ex_9_register_00.PDL",
"Static Text10",PictureName[9]);
}
if (nFlag==REG_DIRECT){
st=1;
pos=nPos;
}
if ((nFlag!=REG_SHOWREGISTER) && (nFlag!=REG_DIRECT)){
if (pos<(biz-1)) SetTagBit("BINi_ex_org_00",FALSE);
else SetTagBit("BINi_ex_org_00",TRUE);

if (pos>0) SetTagBit("BINi_ex_org_01",FALSE);
else SetTagBit("BINi_ex_org_01",TRUE);
}
SetTagWord("U16i_ex_org_00",(WORD)pos);
}

```

- If the transfer parameter *nFlag* has the value *REG_INSERTPICTURE*, the function has been called by the *C-Action* at *Events* → *Property Topics* → *Miscellaneous* → *Picture Name* of the *workspace* object in the *ex_0_startpicture_00.pdl* picture. The *workspace* object is the picture window in which all sample pictures are displayed. If the picture change is not to be entered in the index, the *st* tag must be set to *1* during a previous function call. The index itself consists of a static array with 10 text tags.
- If the transfer parameter *nFlag* has the value *REG_BACK*, the back button has been pressed. The picture change is performed by the function itself, but it is not entered in the index.
- If the transfer parameter *nFlag* has the value *REG_FORWARD*, the next button has been pressed. The picture change is performed by the function itself, but it is not entered in the index.
- If the transfer parameter *nFlag* has the value *REG_SHOWREGISTER*, all *Static Texts* in the *ex_9_register_00.pdl* picture are updated. This is the case, if the index picture is selected or if a picture change is performed while the index picture is open.
- If the transfer parameter *nFlag* has the value *REG_DIRECT*, a direct picture selection via the *Static Texts* has been performed. The picture change is performed by the *C-Action* of the *Static Text*, however, the change is not entered in the index.


Note for the General Application

The following adaptations must be made before the general application:

- The configurations made can directly be applied. For the operation, the 5 tags used must be created, the project function be applied and the control elements be inserted.
- If a direct picture selection and a display of the picture index is not required, the segments *REG_DIRECT* and *REG_SHOWREGISTER* can be omitted from the project function.
- To change the number of pictures stored, the *MAX_REG* definition for the maximum picture number must be changed in the project function.

4.5.2 Index



The index of the sample project is accessed by selecting the button displayed above using the .

Task Definition


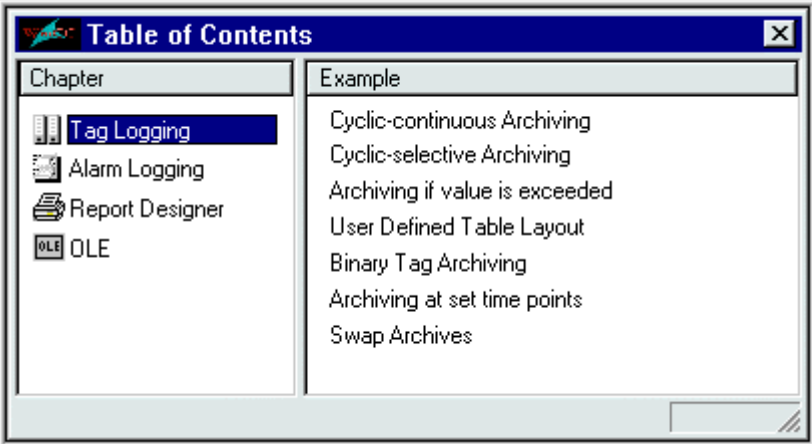
Via a dialog, the index of the project is to be displayed. In one window, the main chapters are to be displayed. In another window, the samples pertaining to a selected chapter are to be displayed.


The direct selection of a sample should be possible. This selection is to be made via a double-click.

Implementation Concept

The selection of the index dialog is performed by a button on the overview bar. This dialog is displayed using a picture window. The dialog contains an additional picture window, which displays a picture depending on the selected chapter.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Two tags of the <i>Unsigned 16-Bit Value</i> type are used. In the sample, these are the <i>U16i_ex_con_00</i> and <i>U16i_ex_con_01</i> tags. These tags store the currently selected chapter number and the sample number.
2	<p>A picture is available, in which the layout of the dialog has been created. This is the <i>ex_9_register_01.pdl</i> picture. For each chapter, a <i>Standard Object</i> → <i>Static Text</i> and a <i>Smart Object</i> → <i>Graphic Object</i> is configured.</p> <p>Initially, at the selection of the picture, no chapter is selected - the <i>U16i_ex_con_00</i> tag has the value zero. If a <i>Static Text</i> is selected with the , the corresponding chapter number is written to this tag. Using various <i>Dynamic Dialogs</i>, the coloring of the selected <i>Static Text</i> is changed.</p> 

Step	Procedure: Implementation in the Graphics Designer
3	<p>For each chapter, a separate picture is configured, which - depending on the chapter number selected - is displayed in a <i>Smart Object</i> → <i>Picture Window</i>. If no chapter is selected, the picture window is not displayed.</p> <p>For each sample (of the chapters), a <i>Standard Object</i> → <i>Static Text</i> is configured. If a chapter is selected, no sample will be displayed initially. If a <i>Static Text</i> is selected with the , the corresponding sample number is written to the <i>U16i_ex_con_01</i> tag. Using various <i>Dynamic Dialogs</i>, the coloring of the selected <i>Static Text</i> is changed.</p>
4	<p>To implement the picture selection via a double-click, three external C variables (listed below) were created. They are created in the project function <i>CreateExternal</i>. This function is performed once at the start of the project.</p> <ul style="list-style-type: none"> • extern BOOL bPress1, bPress2 • extern int nButtonID <p>Via a <i>C-Action</i> at <i>Properties</i> → <i>Geometry</i> → <i>Picture Width</i> of the picture, a query is made in 500 ms cycles whether a double-click has been performed. If this is the case, an action is performed depending on the <i>nButtonID</i> variable.</p>

C-Action at a Sample Text

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
extern BOOL bPress1, bPress2;
static BOOL bToggle = FALSE;
extern int nButtonID;

nButtonID=1;

if (bToggle) bPress1=TRUE;
else bPress2=TRUE;

bToggle=!bToggle;

SetTagWord( "U16i_ex_cont_01", (WORD)nButtonID);
}
```

- The external C variable is supplied with the identification number of the *Static Text*. This identification number is used to determine the action to be performed.
- For every mouse action, *bPress1* and *bPress2* are set alternately to *TRUE*.

C-Action for Determining a Double-Click

```

#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
extern BOOL bPress1,bPress2;
extern int nButtonID;

if ((bPress1)&&(bPress2))
switch(nButtonID){
case1:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01.PDL");
break;
case2:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01a.PDL");
break;
case3:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01b.PDL");
break;
case4:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01c.PDL");
break;
case5:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01d.PDL");
break;
case6:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01e.PDL");
break;
case7:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01f.PDL");
break;
case8:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01g.PDL");
break;
case9:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01i.PDL");
break;
case10:SetPictureName("ex_0_startpicture_00.PDL",
"workspace", "ex_3_chapter_01j.PDL");
break;
default:break;
}

bPress1=FALSE;
bPress2=FALSE;


return 242;
}

```

- Both external C variables are queried. If both have the status *TRUE*, a *Static Text* has been clicked on twice in the last 500 ms. That is, the *C-Action* is performed every 500 ms - after each call, both external C variables are reset to *FALSE*.
- If a double-click is recognized, an action is performed depending on the *nButtonID* variable. This variable contains a reference to which *Static Text* has been selected.
- Return of the picture width.

4.5.3 Color Dialogs (ex_3_chapter_01c)




The color dialogs described in this sample are accessed by selecting the button displayed above using the  in the *ex_3_chapter_01c* picture.

Task Definition

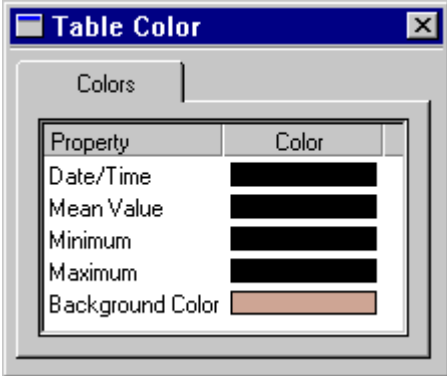
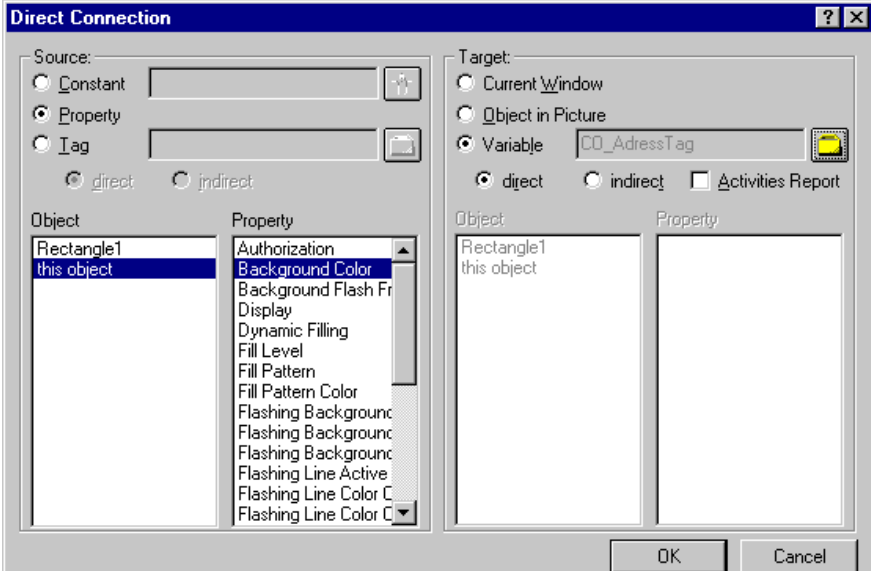
Using various dialog windows, the color settings of the table described in the *Tag Logging* chapter, User-Defined Table Layout (*ex_3_chapter_01c.pdl*) sample, are to be changed. The background color of the table as well as the font color of each column is to be editable.



Implementation Concept


The color dialogs are implemented using a total of three pictures. The first picture, accessed after selecting the button displayed above, contains a dialog displaying the current color settings. From this dialog, a color selection dialog - containing basic 16 colors - can be accessed via a  for each table property that can be set. From this dialog, another color selection dialog - containing 50 colors - can be accessed via a button.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	<p>Five tags of the <i>Unsigned 32-Bit Value</i> type are used. In the sample, these are the <i>CO_TIME</i>, <i>CO_MAX</i>, <i>CO_MIN</i>, <i>CO_MEAN</i> and <i>CO_BACK</i> tags. These tags store the current color values.</p> <p>An additional tag of the <i>Unsigned 32-Bit Value</i> type is used. In the sample, this is the <i>CO_TEMP</i> tag. This tag serves as a buffer for a color value that is to be applied with the <i>OK Button</i>.</p> <p>A tag of the <i>Text Tag 16-Bit Character Set</i> type is used as an address tag. This tag stores the name of the color tag to be processed. In the sample, this is the <i>CO_AdressTag</i>.</p>
2	<p>The table configured in the <i>ex_3_chapter_01c.pdl</i> picture consists of multiple single objects. All properties of these objects - that are to be changed - are equipped with a <i>Tag Connection</i> to one of the color tags.</p>

Step	Procedure: Implementation in the Graphics Designer
3	<p>A picture is available, in which the layout of the first dialog window has been created. This is the <i>ex_10_FD_00.pdl</i> picture. For each property that can be set, a <i>Standard Object</i> → <i>Static Text</i> and a <i>Standard Object</i> → <i>Rectangle</i> is configured.</p> <p>The rectangle displays the currently set color of each property. This is realized via a <i>Tag Connection</i> to the corresponding color tag. Via a <i>C-Action at Events</i> → <i>Mouse</i> → <i>Press left</i> at the <i>Rectangles</i> and the <i>Static Texts</i>, the name of the corresponding color tag is written to the address tag.</p> <p>Via a double-click on one of the rectangles, a dialog for selecting a color from 16 basic colors is opened. The double-click is queried at <i>Properties</i> → <i>Geometry</i> → <i>Position X</i> of the <i>Graphic Object1</i>.</p> 
4	<p>The layout of the second dialog window has been configured in another picture. This is the <i>ex_10_FD_01.pdl</i> picture. For each selectable color, a <i>Standard Object</i> → <i>Rectangle</i> is configured, whose background color corresponds to the color to be set.</p> <p>At <i>Events</i> → <i>Mouse</i> → <i>Press left</i>, a <i>Direct Connection</i> is created for each rectangle. This connection switches the value of the <i>Background Color</i> property of the corresponding object to the color tag contained in the address tag.</p> 

Step	Procedure: Implementation in the Graphics Designer
5	<p>At Events → Mouse → Mouse Action, a <i>Direct Connection</i> is created for each <i>Rectangle</i> that makes the current window invisible.</p> <p>Via a separate <i>Windows Object</i> → <i>Button</i>, the next dialog can be opened.</p> 
6	<p>The layout of the third dialog window has been configured in another picture. This is the <i>ex_10_FD_03.pdl</i> picture. Just as described in step 4, a <i>Standard Object</i> → <i>Rectangle</i> is configured for each color that can be selected. The <i>Direct Connection</i> at Events → Mouse → <i>Press left</i>, however, does not describe the color tag indicated by the address tag, but the temporary color tag <i>CO_TEMP</i>.</p> <p>The value contained in this tag is only written to the color tag to be processed after the OK button has been pressed.</p> <p>In the picture, a <i>Smart Object</i> → <i>Graphic Object</i> has been configured, which displays the color value currently contained in the <i>CO_TEMP</i> tag. In the sample, this is the <i>Selection</i> object. The position of this object is changed upon the selection of a rectangle via a <i>C-Action</i> at Events → Mouse → <i>Press left</i>.</p> 

Step	Procedure: Implementation in the Graphics Designer
7	<p>In the <i>ex_3_chapter_01c.pdl</i> picture, a <i>Smart Object</i> → <i>Picture Window</i> has been configured for each dialog. The <i>Picture Window</i> contained in the first dialog is opened via a <i>Windows Object</i> → <i>Button</i>.</p> 

4.5.4 Bar Graph Display (ex_3_chapter_01e)

The bar graph display described in this sample has been used in the *Tag Logging* chapter, Archiving at Defined Times (ex_3_chapter_01e.pdl) sample.

Task Definition

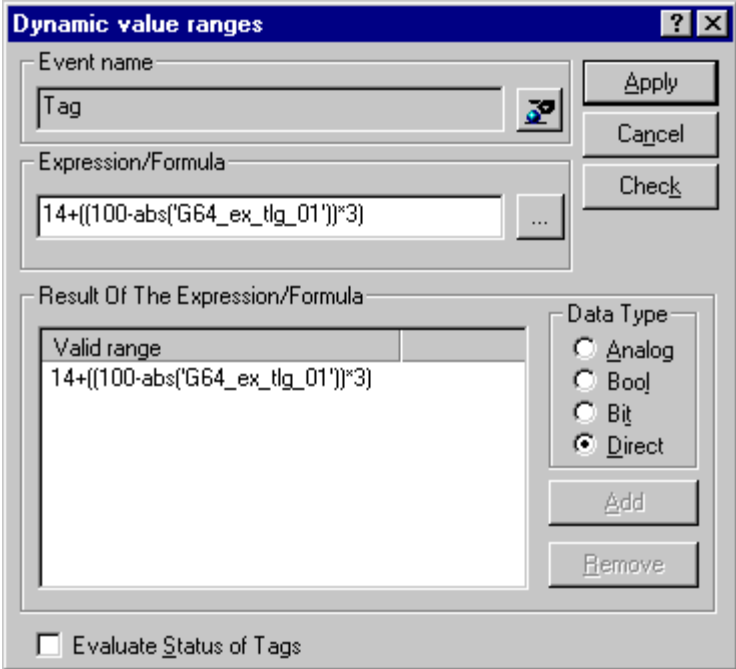

Using three bar graphs, the current values of the tags archived in the corresponding picture are to be displayed. Each bar graph can be activated individually via a button.

Implementation Concept

Each bar graph consists of a *Smart Object* → *Status Display*, which represents the bar graph foreground, and a *Smart Object* → *Picture Window*, in which the background of the bar graph is displayed. Via a *Dynamic Dialog*, the height of the *Picture Window* is controlled depending on the value of the tag to be displayed.

Implementation in the Graphics Designer

Step	Procedure: Implementation in the Graphics Designer
1	Three pictures are available, who only consist of a <i>Smart Object</i> → <i>Graphic Object</i> . These are the <i>ex_10_BH_00.pdl</i> , <i>ex_10_BH_01.pdl</i> and <i>ex_10_BH_02.pdl</i> pictures. In each of the <i>Graphic Objects</i> , bitmaps are displayed representing the bar graph background.
2	In the <i>ex_3_chapter_01e.pdl</i> picture, three <i>Smart Objects</i> → <i>Status Displays</i> are configured, which display the foreground of each bar graph. In case of a deactivated bar graph, they display the background of the bar graph.

Step	Procedure: Implementation in the Graphics Designer
3	<p>Above each <i>Status Display</i>, a <i>Smart Object</i> → <i>Picture Window</i> is placed, with the pictures described in step 1 set at <i>Properties</i> → <i>Miscellaneous</i> → <i>Picture Name</i>.</p> <p>At <i>Properties</i> → <i>Geometry</i> → <i>Window Height</i>, a <i>Dynamic Dialog</i> each is created, which controls the height of the picture window depending on the value of the tag to be displayed.</p> <p>Only positive values can be displayed. Therefore, the absolute value of the tag is formed with the <i>abs()</i> function. The maximum value to be displayed is 100. Since the <i>Picture Window</i> represents the background of the bar graph, the tag value must be subtracted from the maximum value to obtain the background height. One unit of the bar graph in the bitmap equals 3 pixels, therefore the calculated height of the bar graph background must be multiplied by three. Above the bar graph background, 14 pixels are left blank in the bitmap - they must be added to the picture height.</p> 
4	<p>Via three <i>Smart Objects</i> → <i>Status Displays</i> on the toolbar, the bar graphs can be deactivated. This is performed by a <i>C-Action</i> each at <i>Events</i> → <i>Mouse</i> → <i>Press left</i>. This <i>C-Action</i> switches the visibility of the <i>Picture Window</i>, switches the picture containing the bar graph status display and switches the picture displayed by the own object.</p> 

Index

A

- Acknowledgment, 3-57, 4-84
 - Horn, 4-84
 - Messages, 4-84
- Addressing
 - Indirect, 2-52
- Alarm
 - Archiving of, 4-108
 - Bit Message Procedure, 4-71
 - Create, 4-71
 - Limit Value Monitoring, 4-84
 - Line, 3-5
 - Loop in, 4-103
- Analog Values
 - Screen, 4-84
- API, 4-108
 - For Message Filter, 4-108
- Archiving, 4-108
 - Acyclic, 4-27
 - Alarms, 4-108
 - At every full Minute, 4-55
- Area
 - Screen, 3-4
- Authorization Level, 3-40

B

- Bit Message Procedure, 4-71
- Bit Pattern, 2-41
- Bit Processing, 2-40

C

- Callback Function, 4-40
 - Use, 4-40
- Changeover Switch, 2-18
- Color, 3-81
 - Change, 3-70, 3-81
- Configuration
 - Color Scheme of the Messages, 4-71
 - Loop in Alarm, 4-103
- Content
 - From example_01, 4-165
- Control
 - Without a Mouse, 3-88
- Control Windows, 3-52
- Create
 - Group Messages, 4-115

- Message Class, 4-84
- Process Value Archive, 4-3
- Single Messages, 4-71
- Tags, 2-2

D

- Data
 - Archive, 4-3, 4-40
- Decrement, 2-8, 2-20
- Deselection, 3-28
 - Picture, 3-28
 - Picture Window, 3-24, 3-26
- Direct Connection, 2-41, 3-7
- Display, 3-23
 - Picture Window, 3-23
- DLL, 4-89
 - Integrate, 4-89
- Documentation, 4-123
 - Pictures, 4-123
 - Tags, 4-132
 - Trend Windows, 4-137
- Dynamic
 - Part of the Report, 4-148
- Dynamic Dialog, 4-49
 - Use, 4-49
- Dynamic Wizard, 3-11, 3-14

E

- End
 - From WinCC, 3-37
- Export, 2-63
 - Tags, 2-63

G

- Group Messages, 4-115
- Groups
 - Tag Management, 2-2, 2-3
 - User Group, 3-40

H

- Hide
 - Of Information, 3-75, 3-107
- HMI, 3-1
- Hotkey, 3-89
 - Configuration, 3-89

I

- Import, 2-63
 - Tags, 2-63
- Increment, 2-8, 2-20
- Information
 - Display, 3-75, 3-107
 - Hide, 3-75, 3-107
- Information Box, 3-31
 - Configure, 3-31
- Initialization, 4-40
 - In the Project, 3-50
 - Of the Callback Function with Call, 4-40
- Input
 - Check, 3-59, 3-61
 - Via Check-Box, 2-37
 - Via Radio-Button, 2-34

J

- Jog Operation, 2-8

K

- Key Combination, 3-97
 - Logon, Logoff, 3-41
 - Window Switch, 3-97
- Keys
 - For Message Filter, 4-108

L

- Language, 3-86
 - In Runtime, 3-86
- Library, 3-7
 - Project, 3-10
- Login, 3-40, 3-43
- Loop in Alarm, 4-103

M

- Message Windows, 4-71, 4-84, 4-103
- Messages, 4-108
 - Archive, 4-108
 - Bit Message Procedure, 4-71
 - Configure, 4-71
 - Print, 4-153
 - Specify Colors, 4-71
- Multiplex Display, 2-52

O

- OCX
 - Use, 3-110
- Operational
 - Operator-Control Enable, 3-37, 3-40

P

- Password, 3-2, 3-43, 4-27
 - In the Project example_01, 4-27
- Picture, 3-28
 - Change, 3-6
 - Change Size, 3-49
 - Create Picture Index, 4-160
 - Display Picture Name, 3-7
 - Documentation, 4-123
 - Geometry, 3-46
 - Layout, 3-4
 - Project, 3-1
 - Selection, 3-1, 3-7
 - Structure, 3-1
 - Time-Controlled Deselection, 3-28
 - Window, 3-5
- Pragma, 3-42, 4-89
- Print Job, 4-137
 - Time Selection, 4-137
- Process
 - Connection, 2-1
- Process Value Archive, 4-3
 - Create, 4-3
- Project
 - Start, 2-24
- Project Library, 3-10

R

- Resolution
 - Screen, 3-1
- Runtime, 4-18
 - End, 3-38
 - Language Switch, 3-86
 - Print Tables, 4-144
 - Print Trend Windows, 4-137
 - Start, Stop Archiving, 4-18

S

- Screen, 3-3
 - Layout, 3-3
- Set-Point Value, 2-9
 - Change, 2-9, 2-30

- Short-Term Archive, 4-108
 - Create, 4-108
 - Print Messages, 4-153
- Simulation, 2-57, 4-2
- Slider, 2-30
- SmartTools, 2-61
- Sound, 4-89
- Status Display, 3-83
- Structure
 - Tags, 2-5, 2-65
- Switching Operation, 3-53
 - Binary, 3-53
- SysMalloc, 3-110

T

- Table Layout, 4-40
- Tables, 4-40
 - Display Process Values, 4-27
 - User-Defined, 4-40
- Tags, 2-57
 - Archive, 4-49
 - Document, 4-132
 - Simulation, 2-57
- Task, 4-151
 - Message Sequence Report, 4-151

- Text Input, 3-35
- Time, 3-110
 - Displays, 3-110
- Time Selection, 4-137
- Tooltip, 2-3
- Trends, 4-2
 - Display of Process Values, 4-2

U

- Update
 - Trend Windows, 4-18
- User
 - Authorization, 3-40
 - Groups, 3-43

W

- WinCC
 - End, 3-37
- Without a Mouse, 3-88
 - Operation, 3-88
- Wizard
 - For the Process Value Archive, 4-3

