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.

(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

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

Technical data subject to change

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
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Picture Change</td>
<td>3-6</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Opening a Picture via a Direct Connection and Displaying the Picture Name (example 01)</td>
<td>3-7</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Opening a Picture via the Dynamic Wizard (example 02)</td>
<td>3-11</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Opening a Picture via an Internal Function (example 02)</td>
<td>3-13</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Single Picture Change via the Dynamic Wizard (example 03)</td>
<td>3-14</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Single Picture Change via a Direct Connection (example 04)</td>
<td>3-16</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Opening a Picture via the Object Name and an Internal Function (example 05)</td>
<td>3-18</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Opening a Picture via the Object Name and a Tag Connection with Display of the Picture Name (example 06)</td>
<td>3-20</td>
</tr>
<tr>
<td>3.3</td>
<td>Displaying a Picture Window</td>
<td>3-23</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Hiding (Deselection) and Displaying (Selection) from outside the Picture Window (example 01)</td>
<td>3-24</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Displaying (Selection) from outside and Hiding (Deselection) from within the Picture Window (example 02)</td>
<td>3-26</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Time-Controlled Hiding of a Picture (example 03)</td>
<td>3-28</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Displaying a Picture Window while the Right Mouse Button is Pressed (example 04)</td>
<td>3-30</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Configuring Information Boxes with the Wizard (example 05)</td>
<td>3-31</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Displaying a Dialog for Text Input (example 06)</td>
<td>3-35</td>
</tr>
<tr>
<td>3.4</td>
<td>Operator-Control Enable</td>
<td>3-37</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Exiting Runtime and System (example 01)</td>
<td>3-38</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Operator-Control Enable, Logon with Default Box (example 02)</td>
<td>3-40</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Operator-Control Enable, Logon via a separate Dialog (example 03)</td>
<td>3-43</td>
</tr>
<tr>
<td>3.5</td>
<td>Picture Zoom</td>
<td>3-45</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Changing the Picture Geometry between two Sizes (example 01)</td>
<td>3-46</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Changing the Picture Geometry Continuously (example 02)</td>
<td>3-49</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Configuring an adjustable Picture Geometry via the Properties Dialog (example 03)</td>
<td>3-51</td>
</tr>
<tr>
<td>3.6</td>
<td>Control Windows</td>
<td>3-52</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Binary Switching Operation (Two-Step Control) (example 01)</td>
<td>3-53</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Binary S-R Switching Operation (Two-Step Control) (example 02)</td>
<td>3-55</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Binary Switching Operation with Acknowledgment (example 03)</td>
<td>3-57</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Automatic Input Check (example 04)</td>
<td>3-59</td>
</tr>
<tr>
<td>3.6.5</td>
<td>Enhanced Automatic Input Check (example 05)</td>
<td>3-61</td>
</tr>
<tr>
<td>3.6.6</td>
<td>Multiple Operation (example 06)</td>
<td>3-65</td>
</tr>
<tr>
<td>3.7</td>
<td>Dynamization</td>
<td>3-69</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Color Change (example 01)</td>
<td>3-70</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Text Change (example 02)</td>
<td>3-73</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Animation of Movement (example 03)</td>
<td>3-74</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Displaying and Hiding Objects using a Bit Evaluation (example 04)</td>
<td>3-75</td>
</tr>
<tr>
<td>3.7.5</td>
<td>Animation of Movement via a C-Action (example 05)</td>
<td>3-77</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.7.6</td>
<td>Creating Animation of Movement with a Wizard (example 06)</td>
<td>3-79</td>
</tr>
<tr>
<td>3.7.7</td>
<td>Color Change via a C-Action (example 06)</td>
<td>3-81</td>
</tr>
<tr>
<td>3.7.8</td>
<td>Animation of Movement via a Status Display (example 07)</td>
<td>3-83</td>
</tr>
<tr>
<td>3.8</td>
<td>Language Switch</td>
<td>3-85</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Runtime Language Switch (example 01)</td>
<td>3-86</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Dialog Box for the Runtime and Control Center Language Switch (example 02)</td>
<td>3-87</td>
</tr>
<tr>
<td>3.9</td>
<td>Operation without a Mouse</td>
<td>3-88</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Operation via TAB Key or Hotkey (example 01)</td>
<td>3-89</td>
</tr>
<tr>
<td>3.9.2</td>
<td>Cursor Keyboard (example 02)</td>
<td>3-98</td>
</tr>
<tr>
<td>3.9.3</td>
<td>Entering Values, Switching Operations (example 03)</td>
<td>3-103</td>
</tr>
<tr>
<td>3.10</td>
<td>Displaying and Hiding Information</td>
<td>3-107</td>
</tr>
<tr>
<td>3.10.1</td>
<td>Displaying and Hiding Objects (example 01)</td>
<td>3-108</td>
</tr>
<tr>
<td>3.10.2</td>
<td>Date and Time Display (example 02)</td>
<td>3-110</td>
</tr>
<tr>
<td>4</td>
<td>WinCC Editors (Project_WinCCEditors)</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1</td>
<td>Tag Logging</td>
<td>4-2</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Cyclic-Continuous Archiving (ex_3_chapter_01.pdl)</td>
<td>4-3</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Cyclic-Selective Archiving (ex_3_chapter_01a.pdl)</td>
<td>4-18</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Archiving if Values are Exceeded (ex_3_chapter_01b.pdl)</td>
<td>4-27</td>
</tr>
<tr>
<td>4.1.4</td>
<td>User-Defined Table Layout (ex_3_chapter_01c.pdl)</td>
<td>4-40</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Archiving Binary Tags (ex_3_chapter_01d.pdl)</td>
<td>4-49</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Archiving at Defined Times (ex_3_chapter_01e.pdl)</td>
<td>4-56</td>
</tr>
<tr>
<td>4.1.7</td>
<td>Exporting Archives (ex_3_chapter_01f.pdl)</td>
<td>4-62</td>
</tr>
<tr>
<td>4.2</td>
<td>Alarm Logging</td>
<td>4-70</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Bit Message Procedure (ex_3_chapter_02.pdl)</td>
<td>4-71</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Limit Value Monitoring (ex_3_chapter_02a.pdl)</td>
<td>4-84</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Limit Value Monitoring (Continuation)</td>
<td>4-89</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Message Window (ex_3_chapter_02b.pdl)</td>
<td>4-103</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Message Archiving (ex_3_chapter_02c.pdl)</td>
<td>4-108</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Group Messages (ex_8_generator_00.pdl)</td>
<td>4-115</td>
</tr>
<tr>
<td>4.3</td>
<td>Report Designer</td>
<td>4-122</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Picture Documentation (ex_3_chapter_03.pdl)</td>
<td>4-123</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Reporting of the WinCC Explorer (ex_3_chapter_03.pdl)</td>
<td>4-132</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Reporting of Tag Logging CS (ex_3_chapter_03.pdl)</td>
<td>4-135</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Printing Out Trend Windows in Runtime (ex_3_chapter_01a.pdl)</td>
<td>4-137</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Printing Out Tables in Runtime (ex_3_chapter_01c.pdl)</td>
<td>4-144</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Message Sequence Report (ex_3_chapter_02b.pdl)</td>
<td>4-148</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Message Sequence Report on a Line Printer</td>
<td>4-151</td>
</tr>
<tr>
<td>4.3.8</td>
<td>Message Archive Report (ex_3_chapter_02c.pdl)</td>
<td>4-153</td>
</tr>
<tr>
<td>4.4</td>
<td>OLE Communication with EXCEL</td>
<td>4-155</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Reading and Writing Tag Values (ex_3_chapter_04.pdl)</td>
<td>4-156</td>
</tr>
<tr>
<td>4.5</td>
<td>Additional Configurations in the Samples</td>
<td>4-160</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Picture Index .................................................................</td>
<td>4-161</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Index ..................................................................................</td>
<td>4-165</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Color Dialogs (ex_3_chapter_01c) ...........................................</td>
<td>4-168</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Bar Graph Display (ex_3_chapter_01e) .....................................</td>
<td>4-172</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Downloading the Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Downloading the desired project. This is done from the online document by clicking on the following icon: <img src="image" alt="Project Name" /></td>
</tr>
</tbody>
</table>
| 2    | The dialog box *Download File* will be displayed. In this dialog, select the entry *Execute the Program from this Location*. Confirm the dialog by clicking on *OK*.

![File Download](image) You have chosen to download a file from this location.

.../ct_CreatePicture.exe from

What would you like to do with this file?

- [ ] Run this program from its current location
- [ ] Save this program to disk

- [ ] Always ask before opening this type of file

[OK] [Cancel] [More Info]
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Downloading the Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>The dialog <em>Safety Warning</em> will be displayed. Acknowledge this dialog by clicking on <em>Yes</em>.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Safety Warning dialog" /></td>
</tr>
<tr>
<td></td>
<td>Do you want to install and run &quot;..\CreatePicture.exe&quot; from &quot;?</td>
</tr>
<tr>
<td></td>
<td>The publisher cannot be determined due to the problems below:</td>
</tr>
<tr>
<td></td>
<td>Authenticode signature not found.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Yes, No, More Info buttons" /></td>
</tr>
<tr>
<td>4</td>
<td>The <em>WinZip Self-Extractor</em> 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 <em>C:\Configuration_Manual</em>. Start the unzipping process by clicking on the <em>Unzip</em> button.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="WinZip Self-Extractor dialog" /></td>
</tr>
<tr>
<td></td>
<td>Kicken Sie auf &quot;Unzip&quot;, um alle Dateien in Die selbstextrahierende Datei in den angegebenen Ordner zu entpacken.</td>
</tr>
<tr>
<td></td>
<td>Unzip Verzeichnis:</td>
</tr>
<tr>
<td></td>
<td>C:\Configuration_Manual</td>
</tr>
<tr>
<td></td>
<td>Dateien ohne Nachfrage Überschreiben</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Unzip, WinZip starten, Schließen, Info, Hilfe buttons" /></td>
</tr>
<tr>
<td>5</td>
<td>After the unzipping process has completed, a dialog box will confirm the successful unzipping of the files. Acknowledge this dialog by clicking on <em>OK</em>. The <em>WinZip Self-Extractor</em> dialog is closed via the <em>Close</em> button.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="WinZip Self-Extractor dialog" /></td>
</tr>
<tr>
<td></td>
<td>180 Dateien erfolgreich mit Unzip entpackt</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="OK button" /></td>
</tr>
</tbody>
</table>
1.2 Starting Up the Samples (Single-User Projects)

Starting Up the Samples (Single-User Projects)

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Starting Up the Samples (Single-User Projects)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the WinCC Explorer. 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 Start Local Server, the WinCC project is opened.</td>
</tr>
<tr>
<td>2</td>
<td>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 WinCC Explorer via a R on the Computer entry and then selecting Properties from the pop-up menu.</td>
</tr>
</tbody>
</table>
3 The dialog *Computer List Properties* will be opened. The computer list will display all computers pertaining to the project. By clicking on the *Properties* button, the properties dialog of the computer is accessed.

**Computer list properties**

*The computer list includes all computers that are assigned to the current project.*

**Computer List:**

- MY-COMPUTER

*Opens a dialog for editing the selected entry*

[Diagram of the computer list properties dialog with options for select, delete, and properties.]
Step 4

Procedure: Starting Up the Samples (Single-User Projects)

The properties dialog of the computer will be opened. In the General Information tab, replace the computer entered by the local computer.
### Procedure: Starting Up the Samples (Single-User Projects)

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>In the Graphics-Runtime 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 Full Screen and Scroll Bars must be selected from the window attributes field. Exit the dialog by clicking on OK. Exit the computer properties dialog by clicking on OK as well.</td>
</tr>
<tr>
<td>6</td>
<td>Before the project can be activated, it must be reloaded. Close the project via File → Close and then open it again.</td>
</tr>
</tbody>
</table>

**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.

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

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

**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

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

**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 right-click 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.

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

The WinCC Explorer will now display the new driver entry in Tag Management in addition to the Internal Tags.
### Step 3: Procedure: WinCC Tags

Via a Right-Click on the new driver entry, one or several sub-entries, the so-called **Channel Units**, are displayed.

Creation of a connection. This is done by Right-Click on the entry of a **Channel Unit** and then selecting **New Driver Connection** from the pop-up menu.

![Image: Tag Management]

- New Driver Connection...
- System Parameter
- End...
- Paste
- Properties

### Step 4: Procedure: WinCC Tags

In the dialog displayed, assign a name to the connection in the **General Information** tab.

The parameters for the connection can be set by clicking on the **Properties** button.

### Step 5: Procedure: WinCC Tags

Via a Right-Click on the newly added connection entry, tag groups and tags can be added in the manner outlined above.

### Step 6: Procedure: WinCC Tags

When creating **WinCC Tags**, the address and adapt format settings must be defined in addition to the settings required for **Internal Tags**. The address refers to the address of the tag in the PLC.

---

### 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 1: Procedure: Structure Tags

A new structure is created via a Right-Click on the **Structure Type** entry and then selecting **New Structure Type** from the pop-up menu.

![Image: Structure tag]

- New Structure Tag...
- New Structure Type...
- End...
- Paste
### Step 2
**Procedure: Structure Tags**

- In the dialog displayed, give the structure a new name by clicking on the `NewStructure` entry and then selecting `Rename` from the pop-up menu.

### Step 3
- A new structure element can be added via the `New Element` button.

### Step 4
- Via a click on the newly created element, its data type and name can be specified. For each structure element, you must define whether it is an internal or external tag. Clicking on the `OK` button concludes the configuration and creates the structure type.

#### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Last change</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUi_vara_str_00.activated</td>
<td>Binary Tag</td>
<td>Internal tag</td>
<td>07/10/97 02:26:14</td>
</tr>
<tr>
<td>STUi_vara_str_00.open</td>
<td>Binary Tag</td>
<td>Internal tag</td>
<td>07/10/97 02:26:14</td>
</tr>
<tr>
<td>STUi_vara_str_00.closed</td>
<td>Binary Tag</td>
<td>Internal tag</td>
<td>08/21/97 03:45:32</td>
</tr>
<tr>
<td>STUi_vara_str_00.error</td>
<td>Binary Tag</td>
<td>Internal tag</td>
<td>07/10/97 02:26:14</td>
</tr>
</tbody>
</table>
2.2 Incrementing, Decrementing, Jogging

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

Definitions

Incrementing Refers to increasing a tag value by fixed or variable increments.
• Decrementing Refers to decreasing a tag value by fixed or variable increments.
• Jogging Refers 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

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

```c
#include "apdsap.h"
void OnButtonDown(char* lpstrPictureName, char* lpstrObjectName, char* lpstrF
{
    DWORD value = 0;
    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_10", 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

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

```c
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>low) && (v<step)) //low limit
            else v=v-step; //decrement
            //case 0
            break;
    case 1:
        if ((v>high) && (v>step)) //high limit
            else v=v+step; //increment
            //case 1
            break;
    } //switch
    *value=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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a tag of the Signed 32-Bit Value type in Tag Management. In this sample, the S32i_varia_but_04 tag is used.</td>
</tr>
<tr>
<td>2</td>
<td>In a picture, configure the Smart Object ➔ I/O Field. In this sample, the I/O Field2 object is used. During the configuration of the I/O Field in the configuration dialog, the S32i_varia_but_04 variable is set. Change the 2 s default value in the Update field to Upon Change and set the Field Type to Output.</td>
</tr>
<tr>
<td>3</td>
<td>In the same picture, configure a Windows Object ➔ Button. In this sample, the Button7 object is used.</td>
</tr>
<tr>
<td>4</td>
<td>To change the set-point value, create a C-Action at Event ➔ Mouse ➔ Press Left. This C-Action calls the project function <code>IncDecValue</code> and transfers the required parameters to it. This changes the value of the tag each time the button is clicked with the ▼. The limit values are specified as the transfer parameters when the <code>project function</code> is called. The check is performed in the <code>project function</code>.</td>
</tr>
</tbody>
</table>
### Step 5: Procedure: Implementation in the Graphics Designer

Configure the decrementation of the set-point value in the same manner. In this sample, the Button6 object is used.

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

#### C-Action at Button7

```c
#include "apdevag.h"
void cmLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    DWORD value;
    value=GetTagDWord("S321_varia_but_04");
    //IncDecValue(DWORD *value DJORD low DWORD high DWORD step DWORD a )
    IncDecValue(&value,0,100,101,1);
    SetTagDWord("S321_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 \( \text{\textsuperscript{3}} \). 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**

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Direct Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a tag of the Binary Tag type in Tag Management. In this sample, the BINi_varia_but_12 tag is used.</td>
</tr>
<tr>
<td>2</td>
<td>In a picture, we configure the Windows Object → Button. In this sample, the Button2 object is used.</td>
</tr>
</tbody>
</table>
For the Button2, configure a Direct Connection at Event → Mouse → Press Left. Connect the Source Constant → 1 to the Target Tag → BINi_varia_but_12. Apply the settings by clicking on the OK button. Configure another Direct Connection at Event → Mouse → Release Left, but this time for the Source Constant → 0.

The animation is controlled via the BINi_varia_but_12 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

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

```c
#include "epedef.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    SetTagWord("BDi_varis_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

```c
#include "apdefag.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    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(6x0);  //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 \( \sqrt{ } \). 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

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

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

C-Action for the Changeover Switch

```c
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{  
    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:

```c
SetTagDWord("BINi_varia_but_16",
(SHORT)GetTagBit("BINi_varia_but_16");
```
2.2.5 Incrementing and Decrementing (example 01)

The example pertaining to this topic is accessed in the Project_TagHandling project by selecting the Button displayed above using the ﾍ. 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 ﾍ, the value of an internal tag is increased by the increment set, when the button is pressed with the ﾌ, 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Incrementing, Decrementing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creating the tag in the Tag Management. In this sample, the S32i_varia_but_00 and S08i_varia_but_01 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>In a picture, configure a Smart Object → I/O Field. In this sample, the I/O Field1 object is used. During the creation of the I/O Field, set the S32i_varia_but_00 tag in the configuration dialog. Change the 2 s default value in the Update field to Upon Change and set the Field Type to Output.</td>
</tr>
<tr>
<td>3</td>
<td>In the same picture, configure a Windows Object → Button. In this sample, the Button3 object is used.</td>
</tr>
<tr>
<td>4</td>
<td>For the set-point value change via a mouse click, several direct connections are created at this Button. These direct connections change the value of the S08i_varia_but_01 tag each time the Button 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).</td>
</tr>
<tr>
<td>5</td>
<td>The value change of the S32i_varia_but_00 tag is carried out in a C-Action at the</td>
</tr>
</tbody>
</table>
C-Action at the I/O Field for the Value Change

```c
#include "opdefop.h"

long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    DWORD value;
    SHORT count;

    count = GetTagValue("S081_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
}
```

6. The trigger for calling the C-Action is changed to 250 ms.
The declaration of the \textit{C} variables \textit{value} and \textit{count}.

Evaluation of whether the \textit{Button} is pressed. If the \textit{Button} is not pressed, the \textit{C-Action} is ended (to avoid unnecessary system loads).

If the \textit{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 \textit{return}. It is not supposed to be changed.

\textbf{Note for the General Application}

The button with the \textit{direct connections} can be used after changing the tags and in conjunction with the \textit{C-Action} at the \textit{I/O Field}. In the \textit{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 \( \text{ } \), the value of an internal tag is increased by the increment set, when the button is pressed with the \( \text{ } R \), 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 \( \rightarrow \) 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creating the Project Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the WinCC Explorer, start the Global Script editor.</td>
</tr>
<tr>
<td>2</td>
<td>Create a new function via the File ( \rightarrow ) New Project Function menus.</td>
</tr>
<tr>
<td>3</td>
<td>Assign the InitAction function name and save the function by selecting File ( \rightarrow ) Save As ( \rightarrow ) InitAction.fct.</td>
</tr>
<tr>
<td>4</td>
<td>Program and compile the function.</td>
</tr>
</tbody>
</table>
Project Function InitAction

```c
//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

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

Global Action counter.pas

```c
#include "apdsfap.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**

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

**C-Action at Button8 for Incrementation ON**

```c
#include "apdsiap.h"
void UnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    // inc on
    extern char tagsname[301];
    extern SHORT count;
    extern DWORD low;
    extern DWORD high;
    extern DWORD step;

    strcpy(tagsname, "S32i_varia_but_04");
    count = 1;
    low = 0;
    high = 1400;
    step = 1;
}
```
C-Action at Button8 for Incrementation OFF

```c
#include "apdnap.h"
void OnLButtonUp(char* lpszPictureName, char* lpszObjectName, char* lpszProc

{ 
  // 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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a tag in Tag Management. In this sample, the S32i_varia_win_01 tag is used.</td>
</tr>
</tbody>
</table>
### Step 2

In a picture, configure the *Smart Object ➔ I/O Field*. In this sample, the *I/O Field1* object is used. During the creation of the *I/O Field*, set the *S32l_varia_win_01* tag in the *configuration dialog*. Change the 2 s default value in the *Update* field to *Upon Change* and set the *Field Type* to *Output*.

![I/O Field Configuration](image)

### Step 3

In the same picture, configure a *Windows Object ➔ Slider Object*. In this sample, the *Slider Object1* is used. At *Event ➔ Miscellaneous ➔ Process Driver Connection*, create a *direct connection*.

![Object Properties](image)
Step 4

In the Direct Connection dialog, connect the source this object ➔ Process Driver Connection with the target Variable ➔ S32_varia_win_01. Apply the settings by clicking on the OK button.

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

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

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 \textit{Option Group}. When one of the specified set-point values is selecting via a \includegraphics[width=0.05\textwidth]{radio_button}, the value in an \textit{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 \textit{Smart Object \rightarrow I/O Field}. The output value of the \textit{I/O Field} is connected to the \textit{Internal Tag}. The set-point value change is implemented via a \textit{C-Action}.

Implementation in the WinCC project

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

```c
void OnPropertyChange(char* IpszPictureName, char* IpszObjectName, char* Ipsz5

// Set tag according to selected box
switch(value){
    case 1: SetTagDWord("S32i_varia_win_02", 1);
        break;
    case 2: SetTagDWord("S32i_varia_win_02", 50);
        break;
    case 3: SetTagDWord("S32i_varia_win_02", 100);
        break;
    case 4: 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 $\rightarrow$ Check-Box is used, which sets the individual Bits of a tag. A number of Standard Objects $\rightarrow$ Polygons are displayed or hidden, depending on these bits. To display the binary output value of the check-box, a Smart Object $\rightarrow$ I/O Field is used.

Implementation in the WinCC project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Input via a Check-Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a tag of the Signed 32-Bit Value type in Tag Management. In this sample, the S32i_varia_win_03 tag is used.</td>
</tr>
<tr>
<td>2</td>
<td>Configure several Standard Objects $\rightarrow$ Polygons. In this sample, Polygon1 to Polygon7 are used. These objects are to be displayed or hidden depending on the selection status of the Check-Box.</td>
</tr>
<tr>
<td>3</td>
<td>In the same picture, configure a Windows Object $\rightarrow$ Check-Box. In this sample, this is the Check-Box1. At Property $\rightarrow$ Geometry $\rightarrow$ Number of Boxes, change the default value 3 to 7.</td>
</tr>
<tr>
<td>4</td>
<td>Select the Index 1 via Properties $\rightarrow$ Font $\rightarrow$ Index $\rightarrow$ 1. Enter the appropriate text for the selected index at Properties $\rightarrow$ Font $\rightarrow$ Text. 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.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Input via a Check-Box</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>5</td>
<td>At <strong>Events</strong> ➔ <strong>Property Topics</strong> ➔ <strong>Output/Input</strong> ➔ <strong>Selected Boxes</strong>, create a <strong>C-Action</strong>, which assigns the binary status <strong>Check-Box1</strong> to the <strong>S32i_varia_win_03</strong> tag and controls the display of the individual <strong>Polygon</strong> objects.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Object Properties" /></td>
</tr>
<tr>
<td>6</td>
<td>Configure a <strong>Smart Object</strong> ➔ <strong>I/O Field</strong>. In this sample, the <strong>I/O Field4</strong> object is used. In the <strong>configuration dialog</strong>, set the <strong>S32i_varia_win_03</strong> tag. Change the <strong>Update</strong> default value from 2 s to <strong>Upon Change</strong>. At <strong>Properties</strong> ➔ <strong>Output/Input</strong>, change the <strong>Data Format</strong> to <strong>Binary</strong> and the Output Format to <strong>01111111</strong>.</td>
</tr>
</tbody>
</table>
C-Action at the Check-Box

```c
#include "apdctop.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* lpszObjectName, char* lpszObjectName) {
    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 stati. 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 status, 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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>In the same picture, configure a Windows Object ➔ Check-Box. In this sample, the Check-Box1 object is used. At Properties ➔ Geometry ➔ Number of Boxes, change the default value 3 to 16.</td>
</tr>
<tr>
<td>4</td>
<td>Select the Index 1 via Properties ➔ Font ➔ Index ➔ 1. Enter the appropriate text for the selected index at Properties ➔ Font ➔ Text ➔ bit 0. In the same manner, configure the texts for the remaining index entries.</td>
</tr>
<tr>
<td>5</td>
<td>At Event ➔ Property Topics ➔ Selected Boxes, make this event dynamic using a direct connection.</td>
</tr>
</tbody>
</table>

![Object Properties](image)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Setting a Bit directly via a Check-Box and Direct Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>In the <strong>Direct Connection</strong> dialog, connect the <strong>source Property</strong> ➔ <strong>this object</strong> ➔ <strong>Selected Boxes</strong> with the <strong>target Variable</strong> ➔ <strong>U16i_varia_set_05</strong>. Apply the settings by clicking on the <strong>OK</strong> button.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Direct Connection Diagram" /></td>
</tr>
</tbody>
</table>

7 | Configure two **Windows Objects** ➔ **Buttons**. In this sample, the **Button2** and **Button3** objects are used. These will be used to set and reset all bits. |

8 | For **Button2**, create a **direct connection** at **Event** ➔ **Mouse** ➔ **Mouse Action**. Connect the **source Constant** ➔ **65535** with the **target Object in Picture** ➔ **Check-Box1** ➔ **Selected Boxes**. Apply the settings by clicking on the **OK** button. The constant selected corresponds to the binary number 1111111111111111.

For **Button3**, create a **direct connection** in the same manner, but with the **source Constant** ➔ **0**. |

**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 ‘enter bit number’, the selected bit in an internal tag is changed. The change is realized using a C-Action.

![Diagram](image)
**Implementation in the WinCC project**

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

```c
#include "apdefap.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    WORD word, pos;
    //get word and bit position
    pos=GetTagWord("U16i_varia_set_08");
    word=GetTagWord("U16i_varia_set_08");
    word = (WORD)(word1<<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

```c
#include "apdefap.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    WORD word, pos;
    //get word and bit position
    pos=GetTagWord("U16i_varia_set_08");
    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

```c
#include "apdefasp.h"

void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{            
  WORD word, pos;

  // get word and bit position
  pos=GetTagWord("U016i_varia_set_09");
  word=GetTagWord("U016i_varia_set_08");

  word = (WORD)(word`1<<pos);
  SetTagWord("U016i_varia_set_09",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 button. 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

The solutions pertaining to this topic are accessed in the Project_TagHandling project by selecting the Button displayed above using the \sqrt[3]{a}. 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

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

![Object Properties](image)
**Step | Procedure: Indirect Addressing via a Direct Connection**

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

**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**

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

```c
#include "opdefap.h"
void CnPropyChanged(char* lpszPictureName, char* lpszObjectName, char* lpszObjectStatus, char* lpszObjectValue)
{
    char address1[20], address2[20], address3[20];

    switch(value) {
        case 2:
            strcpy(address1, 'S32i_varia_adr_03');
            strcpy(address2, 'S32i_varia_adr_04');
            strcpy(address3, 'S32i_varia_adr_05');
            break;
        case 4:
            strcpy(address1, 'S32i_varia_adr_04');
            strcpy(address2, 'S32i_varia_adr_06');
            strcpy(address3, 'S32i_varia_adr_07');
            break;
        default:
            strcpy(address1, 'S32i_varia_adr_05');
            strcpy(address2, 'S32i_varia_adr_06');
            strcpy(address3, 'S32i_varia_adr_10');
            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

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

```c
#include "apd-tag.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* lpszTagPath, char* lpszVariable, int iValue, char* lpszName, char* lpszType, char* lpszFunction)
{
    char address[40];

    // set tag according to input value
    switch(iValue) {
    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");
    }
    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

element 04

The functionality of this example is to display the process values used in example 02.
2.6 Simulation of Tags

The solutions pertaining to this topic are accessed in the Project_TagHandling project by selecting the Button displayed above using the \( \sqrt{\text{Button}} \). 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Simulation of a Triangular Oscillation via a C-Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create three tags of the Signed 32-Bit Value in Tag Management. In this sample, the S32i_varia_sim_00, S32i_varia_sim_02 and S32i_varia_sim_03 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>Create two tags of the Binary Tag type. In this sample, the BINi_varia_sim_01 and BINi_varia_sim_04 tags are used.</td>
</tr>
<tr>
<td>3</td>
<td>Configure three Smart Objects → I/O Fields. In this sample, the I/O Field1, I/O Field2 and I/O Field3 objects are used.</td>
</tr>
<tr>
<td>4</td>
<td>During the creation of I/O Field1, set the tag S32i_varia_sim_03 and the trigger Upon Change in the configuration dialog. At Properties → Output Format, change the format to 0999. In the same manner, configure the I/O Field2, but set the tag S32i_varia_sim_02.</td>
</tr>
<tr>
<td>5</td>
<td>To check the plausibility of the I/O Field1 object, configure a Tag Connection to the S32i_varia_sim_02 tag at Properties → Limits → High Limit Value. In the same manner, configure the S32i_varia_sim_03 tag as the Low Limit Value of I/O Field2.</td>
</tr>
<tr>
<td>6</td>
<td>During the creation of the I/O Field3 object, set the tag S32i_varia_sim_00, the trigger Upon Change and the Field Type to Output in the configuration dialog. At Properties → Output/Input → Output Format, change the format to 0999.</td>
</tr>
<tr>
<td>7</td>
<td>Configure a Smart Object → Picture Window, in this sample this is the dialog box. Via Properties → Miscellaneous, change the properties Moveable and Border to Yes and the Picture Name to varia_5_window_00. 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.</td>
</tr>
<tr>
<td>8</td>
<td>Configure a Windows Object → Button, in this sample this is the Button2. At Events → Mouse → Press Left, create a direct connection. Connect the source Constant → 1 with the target variable → BINi_varia_sim_04. This Button is used for the initialization.</td>
</tr>
</tbody>
</table>
Step 9: Configure another Windows Object ➔ Button. In this sample, the Button1 is used. At Events ➔ Mouse ➔ Press Left, create a C-Action that negates the status of the BINi_varia_sim_01 tag. At Properties ➔ Geometry ➔ Position X, create a C-Action that executes the tag simulation.

Step 10: To display the simulation status, configure a Smart Object ➔ Status Display. In this sample, the Status Display1 is used. In the configuration dialog, set the tag BINi_varia_sim_01 and trigger Upon Change. Add another status. For the status 0, set the picture glühbirne_2_24Bit.gif and for status 1 the picture glühbirne_1_24Bit.gif.

C-Action for the Tag Simulation

```c
#include "spdefsp.h"
long_main(char* lpzObjectName, char* lpzObjectName, char* lpzPropertyName)
{
    BOOL state;
    static DWORD lowstore = 0;
    static BOOL statestore = 0;
    static DWORD i = 1;
    static DWORD box = 0;
    int high, low;

    //if button init was pressed
    if (GetTagBit("BINi_varia_sim_01")) {
        (1-lowstore);
        (i=1);
        SetTagDWORD("SII1_varia_sim_01",1);
        SetTagDWORD("BINi_varia_sim_01",0);
    }

    //get simulator state
    state=GetTagBit("BINi_varia_sim_01");
    if (state!=statestore) (box=0);
    statestore=state;

    //get limits
    high=GetTagDWORD("SII2_varia_sim_02");
    low=GetTagDWORD("SII3_varia_sim_03");

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

    //if limits different
    if (high==low) {
        box=0;
        //if simulator is activated
        if (state==TRUE) {
            //inc or dec according to direction
            if (i==1) (i=1+1);
            else (i=i-1);
            //set direction
            if (i==high) (i=1);
            if (i==low) (i=1);
            //init simulator if limit overflow
            if ((i=high) && (i==low)){
                (i=low);
                (i=1);
            }
        }
        //set new value
        SetTagDWORD("SII1_varia_sim_00",1);
    }
    //if state
    if (high==low) {
        //set visible message box
        if ((high==low) && (state==1) && (box==0)) {
            box++;
            SetVisible("veria_3_chapter_05.FDL","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

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

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

    if (GetTagBit("BINI_varia_sim_10")) strcpy(program, "\simeng.exe");
    else strcpy(program, "\ simdsv.exe");

    //build path to sindeu
    strcpy(sindeupath, GetTagChar("T16A_varia_sim_07"));
    strcat(sindeupath, program);

    //execute sindeu
    result=ProgramExecute(sindeupath);

    //if not able to execute sindeu set visible message box
    if (result!=0) {
        SetVisible("varia_3_chapter_05.PDF","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 \( \sqrt{ } \). 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 \( \Rightarrow \) 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 \( \Rightarrow \) I/O Fields.

Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Importing / Exporting Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the correct paths to the excel.exe and var_imex.exe programs.</td>
</tr>
<tr>
<td>2</td>
<td>Start the var_exim.exe program by clicking on the Button Imp/Exp during runtime. The program can also be started directly from the Windows Explorer, without runtime having to be active.</td>
</tr>
<tr>
<td>3</td>
<td>Via the Button ( \ldots ), set the path to the Project_TagHandling project and select the Project_TagHandling.mcp file.</td>
</tr>
<tr>
<td>4</td>
<td>Select the Export selection field. The C-Action for calling external programs is described in sample 1.6.2. Then click on Execute ( \Rightarrow ) OK. The export of the tags is now performed. The program generates a file with the extension vex containing information about the tags, a second file with the extension cex containing information about the connections to the PLC and a third file with the extension dex containing information about tags of the Data Structure type.</td>
</tr>
<tr>
<td>5</td>
<td>Start Excel and open the Project_vex.csv file just generated via File ( \Rightarrow ) Open.</td>
</tr>
<tr>
<td>6</td>
<td>To configure 100 tags of the unsigned 16-Bit value type, proceed as follows. The names assigned to these tags are ranging from U16i_varia_impex_00 to U16i_varia_impex_99.</td>
</tr>
<tr>
<td>7</td>
<td>In the first empty line of the first column enter the name U16i_varia_impex_00. Select the cell and move the mouse pointer to the bottom right corner. While keeping the ( \sqrt{ } ) pressed, drag the mouse pointer downward to automatically fill in the remaining 99 cells.</td>
</tr>
<tr>
<td>Step</td>
<td><strong>Procedure: Importing / Exporting Tags</strong></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>8</td>
<td>In the second column, enter a *; in the third column, enter <em>Internal Tag</em>; in the fourth column, enter <em>impexp</em> as the group name; in the fifth column, enter 2 and in the sixth column, enter 5 as the code for an <em>Unsigned 16-Bit Value</em>. In the remaining columns, the value 0 is entered. Fill in the remaining 99 lines automatically.</td>
</tr>
<tr>
<td>9</td>
<td>Open <em>Var_imex.exe</em> again via the task bar and select the <em>Import</em> selection field. Then click on <em>Execute ➔ OK</em>. After completing the import of the tags, exit the program.</td>
</tr>
<tr>
<td>10</td>
<td>100 new tags have now been created in Tag Management.</td>
</tr>
</tbody>
</table>

**Note:**

Runtime does not have to be active to while importing and exporting tags.
2.8 Using Structure Tags

Definition

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Controlling a Valve with a Structure Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define a new structure tag in the WinCC Explorer. At Data Types, select Structure Types via a +R and then New Structure from the pop-up menu.</td>
</tr>
</tbody>
</table>
Procedure: Controlling a Valve with a Structure Tag

2. In the following window, right-click on New Structure and then select Rename from the pop-up menu. In this sample, the name valve is used. Via the New Element button, add a new structure element. Via a right-click on the just created element, set its data type to Bit.

3. Via the Rename button, change the element name to activated and select the internal radio-button. Define further structure elements as follows:

- Valve
  - BIT aktivated
  - BIT open
  - BIT closed
  - BIT error

4. In Tag Management, create a tag of the valve type. In the sample, the STU1_varia_str_00 tag is used. This action creates the following Binary Tags.
## Step 5
Configure two Windows Objects → Buttons, in this sample, the Button1 and Button2 objects are used. At Button1 → Event → Mouse Action → Press Left, create a C-Action that turns the valve on or off. In the same manner, create a C-Action at Button2 that turns the error bit on or off.

## Step 6
For Button1, create a C-Action at Properties → Geometry → Position X that simulates the external processes at the valve.

## Step 7
We then create three different pictures to display the on, off and error status of the valve. In the sample, these pictures contain two Standard Objects → Polygons. 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_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

<table>
<thead>
<tr>
<th>Overview Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plant Representations</td>
</tr>
<tr>
<td>Button Area</td>
</tr>
</tbody>
</table>

Layout 2

<table>
<thead>
<tr>
<th>Logo</th>
<th>Overview Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button Area</td>
<td>Plant Representations</td>
</tr>
</tbody>
</table>
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

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.pdf and pictu_3_chapter_01a.pdf 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**

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Via File ➔ New, create a new picture and via File ➔ Save As..., save it under the name pictu_5_window_00.pdl. Set Properties ➔ Geometry ➔ Width to 270 and Properties ➔ Geometry ➔ Height to 280.</td>
</tr>
<tr>
<td>2</td>
<td>In the pictu_5_window_00.pdl picture, configure a Standard Object ➔ Static Text. In this sample, the Static Text1 object is used. Set Properties ➔ Font ➔ Bold to Yes. At Properties ➔ Font ➔ Text, delete the default text from the Static column. This prevents an incorrect text from being output at the moment of the picture generation. Make the object at Properties ➔ Font ➔ Text dynamic using a C-Action. This C-Action returns the current picture name as the return value. As the trigger for the C-Action, the Default Cycle ➔ 1 h is used (low system load, no change).</td>
</tr>
</tbody>
</table>

Event Name: 1 h
### Step 3

In the `pictu_5_window_00.pdl` picture, configure the information to be displayed. In the sample, the `tank3` object from the Global Library is used. The library is accessed via `View ➞ Library` or via the `Library` button on the toolbar.

Make sure that the symbol view has been selected via the `Symbol View` button to obtain a preview of the individual objects.

![Library](image1)

### Step 4

Configure two more sample pictures for the picture change by saving the picture you have just configured via `File ➞ Save As...` under the name `pictu_5_window_01.pdl` and this picture again under the name `pictu_5_window_02.pdl`. This gives us two copies of `pictu_5_window_00.pdl`. Now insert the desired content into the new pictures created. There is no need to change the `Static Text1` object for displaying the picture name.

### Step 5

Configure a new picture via `File ➞ New`. Configure a `Smart Object ➞ Picture Window` in this picture. In this sample, the `Picture Window1` object is used. Adapt the dimension of the `Picture Window` via `Properties ➞ Geometry ➞ Width` and `Properties ➞ Geometry ➞ Height` to the size of the previously created pictures. To have the window displayed with a border in runtime, set the `Property ➞ Miscellaneous ➞ Border` to `Yes`. 

![Diagram](image2)
Step | Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name
--- | ---
6 | At Properties → Miscellaneous → Picture Name, set the `pictu_5_window_00.pdl` picture. This sets the picture to be displayed by the Picture Window1 object when the picture is opened.

7 | In the same picture, configure a Windows Object → Button. In this sample, this is the Button1 object. At Events → Mouse → Press Left, create a direct connection.

Select Constant as the Source and click on the now enabled button to display a selection list of all available pictures. Select the picture `pictu_5_window_00.pdl` and the target Object in Picture, the object Picture Window1 and the object property Picture Name.

Apply the settings by clicking on the OK button.
Step | Procedure: Opening a Picture via a Direct Connection and Displaying the Picture Name
--- | ---
8 | Use the to select the configured Button1 object and duplicate is via Edit Duplicate. Repeat this procedure one more time. We now have two more buttons, Button2 and Button3. At Events Mouse Press Left, change the configured direct connection. For Button2, set the source to pictu_5_window_01.pdl and for Button3 to pictu_5_window_02.pdl.

C-Action at Static Text1

```c
#include 'apdisapp.h'
char* _asim(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
char* name = lpszPictureName;
char* pdest;
int ch = '\';

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

- Declare the C variables.
- Check whether lpszPictureName only contains the picture name. This is done via the strchr 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 R. We will use the pictures already configured in the previous example.

Implementation in the WinCC Project

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

Set options

Your dynamic requires additional parameters:

Specify the picture in which a picture window should be changed:

pictu_3_chapter_01.PDL

Select the window object:
Pictu Window2

Please specify the destination picture name:
pictu_5_window_00.PDL
Step | Procedure: Opening a Picture via the Dynamic Wizard
--- | ---
| Via the Button button, the pictures available to the project are accessed. Confirm the Finished! page by clicking on the Finish button. |
| 4 | Configure two additional Windows Objects → Buttons. In this sample, these are the Button5 and Button6 objects. Apply the Dynamic Wizard to these buttons as well. In the Set Options page for Button5, set pictu_5_window_01.pdl as the Destination Picture Name, and pictu_5_window_02.pdl for Button6. |

C-Action generated by the Dynamic Wizard

```c
#include "apdevap.h"
void OnCnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    static char szPicture[22] = 'pictu_5_window_00.PDL';
    static char* lpszObject[10];
    PDLRTSetPropEx(PDLRT_AM_PICTABS, "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

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

#### C-Action at Button4

```c
#include "apdsfap.h"
void OnLButtonDown(char* 1pszPictureName, char* 1pszObjectName, char* 1pszPos)
{
    SetPictureName("pictu3 chapter_01.PDL", "Picture Window2", "pictu5_window_00.pdl");
}
```

- Via the internal function SetPictureName, switch the pictu5_window_00.pdl picture into the Picture Window2 object. pictu3 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Single Picture Change via the Dynamic Wizard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In this sample, the picture change is performed from the <code>pictu_0_startpicture_00.pdl</code> to the <code>pictu_3_chapter_01a.pdl</code> picture. In the sample project, the <code>pictu_0_startpicture_00.pdl</code> picture is always selected and from there, picture changes are only performed in windows. By using the C-Action generated by the Dynamic-Wizard, the entire picture system displayed in runtime is replaced by the one called. Changing back to <code>pictu_0_startpicture_00.pdl</code> is comparable to completely restarting the picture project.</td>
</tr>
<tr>
<td>2</td>
<td>In the picture, configure a Windows Object ( \rightarrow ) Button. In this sample, the <code>Button7</code> object is used.</td>
</tr>
<tr>
<td>3</td>
<td>While the object is highlighted, select the Picture Functions tab and then the Single Picture Change entry via a ( ) from the Dynamic Wizard. From the Select Trigger page of the Dynamic Wizard, select the Left Mouse Button list entry and continue to the next page by clicking on the Next button. Complete the Set Options page as follows:</td>
</tr>
<tr>
<td></td>
<td>Via the Button, a list of all available pictures in the project will be displayed. Confirm the Finished! page by clicking on the Finish button.</td>
</tr>
<tr>
<td>4</td>
<td>If the picture change is performed in the sample project, click on the button ( ) to return to the project.</td>
</tr>
</tbody>
</table>
C-Action generated by the Dynamic Wizard

```c
#include "opcapi.h"
void OnButton(}char* lpszPictureName, char* lpszObjectName, char* lpszErrorName)
{
    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

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

#### 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 is 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 mouse. 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**

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

#include <apdofap.h>
void OnButtonDown(char *pszPictureName, char *pszObjectName, char *pszF

    { 
    char name[30];
    int number;
    char *pdest;

    //check if object name contains character
    pdest = strchr(1pszObjectName, 'n');
    if ( pdest == NULL ) { printf("ObjectNameError");
    else {
        //read object number
        number = atoi(strchr(1pszObjectName, 'n')+1);
        //generate picture name
        sprintf(name,"picture_5_window_%02d.PDL",number);
        //set picture name
        SetPictureName("picture_5_a.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 strchr 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 is 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 \( \sqrt{ } \). 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

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

6 | In the picture, configure a *Smart Object* ➔ *Static Text* above the *Picture Window Picture Window2*. In this sample, the *Static Text4* object is used. Set *Properties ➔ Font ➔ Bold* to Yes. At *Properties ➔ Font ➔ Text*, delete the entered text from the *Static* column and create a *tag connection* to the *T16x_selec_00* tag. Set the update to *Upon Change*. Deleting the static entry prevents an incorrect text from being output at the moment of picture buildup.

![Object Properties](image)

### C-Action at Open Picture

```c
#include "apdstruc.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPath:
{
    SetTagChar("T16x_selec_00", "pic_window_01 pd1");
}
```

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

WinCC Configuration Manual
C79000-G8276-C158-01
3-21
C-Action at Button_0

```c
#include "apdsiap.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    char name[30];
    int number;
    int ch = '_';
    char *pdest;
    //check if object name contains character
    pdest = strchr(lpszObjectName, ch);
    if ( pdest == NULL ) (printf("ObjectNameError\n");
    else {
        //read object number
        number = atoi(strchr(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 `strchr` 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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Hiding and Displaying from outside the Picture Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Configure a picture that is to be displayed and hidden, e.g. a help text or an information box. In the sample, the <em>pictu_5_window_07</em> is used, a pure information box without additional control elements.</td>
</tr>
<tr>
<td>2</td>
<td>In another picture, configure a <em>Smart Object</em> → <em>Picture Window</em> with the same geometric dimension as the previously created picture. In this sample, the <em>Picture Window1</em> object is used. Set the <em>Property</em> → <em>Geometry</em> → <em>Width</em> to 246 and the <em>Property</em> → <em>Geometry</em> → <em>Height</em> to 129. To have the window displayed with a border in runtime, set the <em>Property</em> → <em>Miscellaneous</em> → <em>Border</em> to Yes. To allow the window to be moved around, set the <em>Property</em> → <em>Miscellaneous</em> → <em>Moveable</em> to Yes. To hide the window in runtime, set the <em>Property</em> → <em>Miscellaneous</em> → <em>Display</em> to No. At <em>Properties</em> → <em>Miscellaneous</em> → <em>Picture Name</em>, select the <em>pictu_5_window_07.pdl</em> picture.</td>
</tr>
</tbody>
</table>
### Step 3

In the same picture, configure two additional **Windows Objects** ➔ **Buttons**. In this sample, these are the **Button1** and **Button2** objects. For **Button1**, configure a **direct connection** at **Events** ➔ **Mouse** ➔ **Press Left**. Connect the source **Constant** ➔ 1 with the target **Object in Picture** ➔ **Picture Window1** ➔ **Display**. Apply the settings by clicking on the **OK** button.

#### 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 Press Left.

Implementation in the WinCC Project

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

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

```c
#include "apdefs.h"
int _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    static int i = 0;
    // count time
    ++i;
    // if maximum time is reached
    if (i>5) SetVisible("pictu_5_chapter_03 PDL"."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

```c
#include "apdefs.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{
    // set visibility in complement state
    SetVisible(lpszPictureName,"Picture Window3":
    |
    [SHORT]!GetVisible(lpszPictureName,"Picture Window3")
}
```

- The visibility of the *Picture Window* 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 right mouse button and be hidden again if the right mouse button 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 it is being pressed with the right mouse button.

**Implementation in the WinCC Project**

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

**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 $\rightarrow$ Slider Object to enter the tag value and a Smart Object $\rightarrow$ I/O Field to display the tag value.
Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuring Information Boxes with the Wizard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If the <em>Dynamic Wizard</em> is not displayed, activate it from <em>View ➔ Toolbars</em>.</td>
</tr>
<tr>
<td>2</td>
<td>In a picture, configure a <em>Smart Object ➔ I/O Field</em>. In this sample, the <em>I/O Field1</em> object is used. While the object is highlighted, select the <em>Picture Functions</em> tab and then the <em>Display Instruction Box</em> entry via a D from the Dynamic Wizard. From the <em>Select Trigger</em> page of the <em>Dynamic Wizard</em>, select the <em>Left Mouse Button</em> list entry and continue to the next page by clicking on the <em>Next</em> button. Complete the <em>Set Options</em> page as follows:</td>
</tr>
</tbody>
</table>

![Set options](image)

Confirm the *Finished!* page by clicking on the *Finish* button. |
| 3    | Use the *Dynamic Wizard* again for the *I/O Field1*. On the *Select Trigger* page, select *Right Mouse Button*; on the *Set Options* page, select the *Emergency Box* radio-button and enter the text for the display. |
| 4    | In *Tag Management*, create a tag of the *Signed 32-Bit Value* type. In this sample, the *S32i_pictu_boxes_00* tag is used. |
| 5    | In the same picture, configure a *Windows Object ➔ Slider Object*. In this sample, this is the *Slider Object1*. For the *Slider Object1*, create a direct connection at *Events ➔ Property Topics ➔ Miscellaneous ➔ Process Driver Connection*. Connect the source *Property ➔ Slider Object1 ➔ Process Driver Connection* with the *Target Variable ➔ S32i_pictu_boxes_00*. Apply the settings by clicking on the *OK* button. |
### Step 6

For the I/O Field1 object, create a Tag Connection at Properties ➔ Output/Input ➔ Output Value to the tag `S32i_pictu_boxes_00` and trigger upon change.

<table>
<thead>
<tr>
<th>Property</th>
<th>Output Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Format</td>
<td>Decimal</td>
</tr>
<tr>
<td>Output Format</td>
<td>999,999</td>
</tr>
<tr>
<td>Apply on Full</td>
<td>No</td>
</tr>
<tr>
<td>Apply on Exit</td>
<td>No</td>
</tr>
<tr>
<td>Clear on New Inv</td>
<td>No</td>
</tr>
<tr>
<td>Clear on Inv</td>
<td>No</td>
</tr>
<tr>
<td>Clear on Exit</td>
<td>No</td>
</tr>
<tr>
<td>Clear on Exit</td>
<td>No</td>
</tr>
</tbody>
</table>

### Step 7

For the I/O Field1 object, create a C-Action at Events ➔ Property Topics ➔ Output/Input ➔ Output Value that displays an information box if the `S32i_pictu_boxes_00` tag exceeds the value 100 and an emergency box if the value 150 is exceeded. The C-Actions generated by the Dynamic Wizard at Events ➔ Mouse ➔ Press Left and Press Right can be copied and pasted into this C-Action.

### Step 8

Delete the C-Actions at Events ➔ Mouse ➔ Press Left and Press Right.
Picture Configuration (Project_CreatePicture) 09.99

C-Action at I/O Field1

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

  // get tag value
  a = GetTagDWord("S321_picture_boxes_00");

  // set visible info box
  if (a>100) && (i==0)) {
    MessageBox(NULL, "Der Variable:wert hat\r\n 100 überschritten ", "Info");
    MB_OK|MB_ICONEXCLAMATION|MB_SETFOREGROUND);
  } // if
  if (a<-100) (i=0);

  // set visible emergency box
  if (a>150) && (j==0)) {
    MessageBox(NULL, "Der Variable:wert hat\r\n 150 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Displaying a Dialog for Text Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create two tags of the Text Tag 16-Bit Character Set type. In this sample, the T16i_pictu_win_00 and T16i_pictu_win_01 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>Configure a picture in which text input is to be carried out. In the sample, the pictu_5_window_17.pdl picture is used.</td>
</tr>
<tr>
<td>3</td>
<td>In this picture, configure a Smart Object I/O Field. In its configuration dialog, select the T16i_pictu_win_01 tag and set the trigger to Upon Change. Set the Property Output/Input Data Format to String and the Property Output/Input Apply on Exit to Yes. This means that the ENTER key does not have to be pressed to accept the text entered.</td>
</tr>
<tr>
<td>4</td>
<td>In the same picture, configure a Windows Object Button. In this sample, the Button1 object is used. This button is used to apply the text entered. At Events Mouse Press Left, configure a direct connection with the source Variable T16i_pictu_win_01 and the target Variable T16i_pictu_win_00. At Events Mouse Mouse Action, configure a direct connection that hides the picture.</td>
</tr>
<tr>
<td>5</td>
<td>Configure another Windows Object Button; in this sample, this is the Button2 object. This button is used to cancel the input, the text entered previously is retained. At Events Mouse Press Left, configure a direct connection with the source Variable T16i_pictu_win_00 and the target Variable T16i_pictu_win_01. This direct connection transfers the content of T16i_pictu_win_00 (containing the previous text) to T16i_pictu_win_01. At Events Mouse Mouse Action, configure a direct connection that hides the picture.</td>
</tr>
</tbody>
</table>
### Step 6

In a second picture, configure a *Smart Object* → *Picture Window*. In this sample, the *Picture Window1* object is used. Adjust the dimension of the *Picture Window* to match the size of the picture just created. If the *Picture Window* is to be displayed with a border, the *Height* and *Width* of the *Picture Window* must be 10 pixels greater than those of the picture. At *Properties* → *Miscellaneous* → *Picture Name*, enter `pictu_5_window_17.pdl`.

### Step 7

In the same picture, configure a *Windows Object* → *Button*. In the sample, this is the *Button1* object. At *Events* → *Mouse* → *Press Left*, create a direct connection. Connect the *source Constant* → 1 with the target *Object in Picture* → *Picture Window1* → *Display*. Apply the settings by clicking on the *OK* button.

### Step 8

In the same picture, configure a *Standard Object* → *Static Text*. In the sample, this is the *Static Text1* object. At *Properties* → *Font* → *Text*, configure a *tag connection* to the `T16i_pictu_win_00` 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

The samples pertaining to this topic are accessed in the Project_CreatePicture project by selecting the Button displayed above using the Button. 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 two 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Exiting Runtime and System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Configure a picture, which is going to be used to exit runtime. In the sample, the pictu_5_window_04.pdl picture is used.</td>
</tr>
<tr>
<td>2</td>
<td>In this picture, configure a Windows Object → Button; in the sample, the Button1 object is used. While the object is highlighted, select the System Functions tab and then the Exit WinCC or Windows entry via a ⬇️D from the Dynamic Wizard. From the Select Trigger page of the Dynamic Wizard, select the Left Mouse Button entry and continue to the next page by clicking on the ⬇️Next button. On the Set Options page, select Exit Windows. Confirm the Finished! page by clicking on the Finish button.</td>
</tr>
<tr>
<td>3</td>
<td>Configure another Windows Object → Button. In this sample, the Button2 object is used. This button is used to cancel the procedure. At Events → Mouse → Press Left, configure a direct connection that will hide the picture.</td>
</tr>
<tr>
<td>4</td>
<td>Configure another picture, which is going to be used to shut down the system. In the sample, the pictu_5_window_03.pdl picture is used.</td>
</tr>
<tr>
<td>5</td>
<td>In this picture, configure a Windows Object → Button. In this sample, the Button1 is used. While the object is highlighted, select the System Functions tab and then the Exit WinCC Runtime entry via a ⬇️D from the Dynamic Wizard. From the Select Trigger page of the Dynamic Wizard, select the Left Mouse Button entry and continue to the next page by clicking on the ⬇️Next button. Confirm the Finished! page by clicking on the Finish button.</td>
</tr>
<tr>
<td>6</td>
<td>Configure another Windows Object → Button. In this sample, the Button2 object is used. This button is used to cancel the procedure. At Events → Mouse → Press Left, configure a direct connection that will hide the picture.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Exiting Runtime and System</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>In another picture, configure two <em>Smart Objects</em> ➔ <em>Picture Windows</em>; in this picture, the <em>Picture Window1</em> and <em>Picture Window2</em> object are used that are arranged one on top of the other. Adjust the dimension of the <em>Picture Windows</em> to match the size of the pictures just created. If the <em>Picture Windows</em> are to be displayed with borders, the <em>Height</em> and the <em>Width</em> of the <em>Picture Windows</em> must be set 10 pixels greater than those of the pictures, in order to display the entire picture. At <em>Properties ➔ Miscellaneous ➔ Picture Name</em>, enter the respective picture names. Set the <em>Property ➔ Miscellaneous ➔ Display</em> to <em>No</em>.</td>
</tr>
<tr>
<td>8</td>
<td>In the same picture, configure two <em>Windows Objects ➔ Buttons</em>. In this sample, these are the <em>Button1</em> and <em>Button2</em> objects. For <em>Button1</em>, create a <em>direct connection at Events ➔ Mouse ➔ Press Left</em>. Connect the <em>source Constant ➔ 1</em> with the <em>target Object in Picture ➔ Picture Window1 ➔ Display</em>. Apply the settings by clicking on the <em>OK</em> button. In the same manner, create a <em>direct connection</em> for <em>Button2</em> *<strong>, but set as target Object in Picture ➔ Picture Window2 ➔ Display.</strong></td>
</tr>
</tbody>
</table>

**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 \(\rightarrow\) Buttons, which each display a different picture in a Smart Object \(\rightarrow\) Picture Window when pressed with the \(\sqrt{\text{D}}\). The settings required for assigning user authorizations are made in the User Administrator editor.

Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Operator-Control Enable, Logon with Default Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the WinCC Explorer, open the User Administrator editor by (\sqrt{\text{R}}) on it and then selecting Open from the pop-up menu.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="User Administrator" /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Open" /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Find..." /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Properties" /></td>
</tr>
<tr>
<td>2</td>
<td>Via the button, create a new user group and assign a name to it; in this sample, we will use the name service.</td>
</tr>
<tr>
<td>3</td>
<td>Via the Table (\rightarrow) Add new Authorization Level menus, define the authorization level Picture Change as line number 9. This authorization level is assigned to the service group. To do so, select the group with the (\sqrt{\text{D}}). In the table containing the line Picture Change, select the radio-button in the Authorization column with a (\sqrt{\text{D}}). An authorization level assigned to a group or user is marked by a red dot next to it in the Authorization column.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="9 Picturechange" /></td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Operator-Control Enable, Logon with Default Box</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>Via the Button, create a new user for the service user group. In the sample project, a user named willi with the password Project_CreatePicture has been created. Activate the Copy Group Settings Also check-box to transfer the authorization levels valid for the group to the user.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Establish new user" /></td>
</tr>
<tr>
<td></td>
<td>Login: willi</td>
</tr>
<tr>
<td></td>
<td>Password: **********</td>
</tr>
<tr>
<td></td>
<td>Verify password: **********</td>
</tr>
<tr>
<td></td>
<td>Copy group settings also</td>
</tr>
<tr>
<td></td>
<td>Via the File menu, close the User Administrator editor.</td>
</tr>
<tr>
<td>5</td>
<td>In the WinCC Explorer, right-click on the project name to access the project properties dialog. In the window displayed, select the Hotkeys tab and make the desired settings for calling the Logon and Logoff dialogs. To assign the hotkeys, use the right-click and click on the Assign button. In this sample, the key combination CTRL+O for Logon and CTRL+F for Logoff is used.</td>
</tr>
<tr>
<td></td>
<td>Hot keys are key combinations to which you can assign specific actions within WinCC.</td>
</tr>
<tr>
<td></td>
<td>Actions:</td>
</tr>
<tr>
<td></td>
<td>Logon</td>
</tr>
<tr>
<td></td>
<td>Logoff</td>
</tr>
<tr>
<td></td>
<td>Hardcopy</td>
</tr>
<tr>
<td></td>
<td>Logon: STRG + O</td>
</tr>
<tr>
<td></td>
<td>Logoff: Previous assigned</td>
</tr>
<tr>
<td></td>
<td>Click here to assign the specified shortcut to the action.</td>
</tr>
</tbody>
</table>
### Step 6

**Procedure: Operator-Control Enable, Logon with Default Box**

6. In a picture, configure two *Windows Objects* → *Buttons*. In this sample, the *Button3* and *Button4* objects are used. Configure a *Smart Object* → *Picture Window*, into which pictures are inserted via *direct connections* at the two *Buttons*.

### Step 7

For the *Button3* and *Button4* objects, select the *Picture Change* user level at *Properties* → *Miscellaneous* → *User Level* and set the *Property* → *Miscellaneous* → *Operator-Control Enable* to *No*.

### Step 8

While the *Button3* object is highlighted, select the *Standard Dynamics* tab and then the *Operational if Authorized* entry via a ➞ D from the Dynamic Wizard. Complete the *Dynamic Wizard* by clicking on the *Finish* button. Repeat the same procedure for *Button4*.

### Step 9

In Tag Management, create the @*CurrentUser* system tag of the *Text Tag 16-Bit Character Set* type. The user name currently logged in is automatically assigned to this tag.

### Step 10

Trigger the *C-Actions* at *Button3* and *Button4* generated by the *Dynamic Wizard* upon change of this tag. This means that the *C-Action* will no longer be executed every 2 seconds, but only if the user name changes.

**C-Action generated by the Dynamic Wizard**

```c
#include "apdapap.h"
BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName

{ ...
```

**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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Operator-Control Enable, Logon via a separate Dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the User Administrator 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 ulrich with the password Project_CreatePicture has been created.</td>
</tr>
<tr>
<td>2</td>
<td>In a picture, configure two Windows Objects → Buttons. In this sample, the Button5 and Button6 objects are used.</td>
</tr>
<tr>
<td>3</td>
<td>For Button5, configure the call of a Smart Object → Picture Window for ending runtime. In this sample, the Picture Window Picture Window5 object is used.</td>
</tr>
<tr>
<td>4</td>
<td>For the Button5 object, select the Exiting Runtime user level at Properties → Miscellaneous → User Level and set the Property → Miscellaneous → Operator-Control Enable to No.</td>
</tr>
<tr>
<td>5</td>
<td>Apply the Operational if Authorized Dynamic Wizard to Button5. Set the C-Action generated to be triggered by the @CurrentUser system tag.</td>
</tr>
</tbody>
</table>
Step | Procedure: Operator-Control Enable, Logon via a separate Dialog
--- | ---
6 | Configure a *Smart Object* ➔ *Picture Window*. In this sample, the *Picture Window* object is used. Set the Property ➔ *Geometry* ➔ *Window Height* to 360 and the Property ➔ *Geometry* ➔ *Window Height* to 180. Set the Properties ➔ *Miscellaneous* ➔ Moveable, Border, Title and Foreground to Yes. At Properties ➔ *Miscellaneous* ➔ *Picture Name*, select the *pictu_5_window_18.pdl* picture. This picture can be taken directly from the *Project_CreatePicture* project.

![Login and Password input dialog](image)

7 | For the *Button6* object, create a *direct connection* for displaying the just configured Picture Window.

8 | At the *Button6* object, configure a *C-Action* that assigns a text to the button label depending whether the user is logged on or not. This *C-Action* is also triggered by the *@CurrentUser* tag.

### C-Action at Button6

```c
#include "apdefap.h"
char _main[CHAR 1pszPictureName, char 1pszObjectName, char 1pszProperty
{
  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

The samples pertaining to this topic are accessed in the Project_CreatePicture project by selecting the Button displayed above using the \( \text{Zoom} \). 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

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

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

### C-Action at Button3

```c
#include "apdevap.h"
void OnLButtonDown(char* lpszObjectName, char* lpszObjectName, char* lpszObjectName)
{
SetHeight(lpszObjectName, "Picture Window1", 420);
SetWidth(lpszObjectName, "Picture Window1", 516);
SetVisible(lpszObjectName, "Button3", 0);
SetVisible(lpszObjectName, "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

```c
#include "apdevap.h"
void OnLButtonDown(char* lpszObjectName, char* lpszObjectName, char* lpszObjectName)
{
SetHeight(lpszObjectName, "Picture Window1", 140);
SetWidth(lpszObjectName, "Picture Window1", 172);
SetVisible(lpszObjectName, "Button3", 1);
SetVisible(lpszObjectName, "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

```c
#include "apdsiap.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Changing the Picture Geometry Continuously</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Configure a picture that is to be displayed and hidden. In the sample, the <em>pictu_5_window_10.pdl</em> picture is used, whose width to height ratio is 2 to 1.</td>
</tr>
<tr>
<td>2</td>
<td>In another picture, configure a Smart Object → Picture Window; in the sample, this is the <em>Picture Window2</em>. Set the Property → Geometry → Width to 160 and the Property → Geometry → Height to 80 (width : height ratio is also 2 : 1). To have the window displayed with a border during runtime, set the Property → Miscellaneous → Border to Yes and the Properties → Miscellaneous → Adapt Picture to Yes. In this way, the picture is adapted to the size of the Picture Window. At Properties → Miscellaneous → Picture Name, select the <em>pictu_5_window_10.pdl</em> picture. Set the Property → Miscellaneous → Display to No.</td>
</tr>
<tr>
<td>3</td>
<td>In the same picture, configure two additional → Buttons. In the sample, these are the <em>Button5</em> and <em>Button6</em> objects. For <em>Button5</em>, create a direct connection at Events → Mouse → Press Left. Connect the source Constant → 1 with the target Object in Picture → Picture Window2 → Display. Apply the settings by clicking on the OK button.</td>
</tr>
<tr>
<td>4</td>
<td>In the same manner, create a direct connection for <em>Button6</em> at Events → Mouse → Press Left. As Constant, specify the value 0.</td>
</tr>
<tr>
<td>5</td>
<td>In Tag Management, create a tag of the Unsigned 16-Bit Value type. In this sample, the <em>U16i_pictu_zoom_00</em> tag is used.</td>
</tr>
<tr>
<td>6</td>
<td>Configure a Windows Object → Slider Object. In this sample, this is the <em>Slider Object1</em>. Set the Property → Miscellaneous → Maximum Value to 300. Set the Property → Miscellaneous → Process Driver Connection to 80. At Events → Property Topics → Miscellaneous → Process Driver Connection, create a direct connection. Connect the source Property → this object → Process Driver Connection with the target Variable → <em>U16i_pictu_zoom_00</em>. Apply the settings by clicking on the OK button.</td>
</tr>
</tbody>
</table>
Step | Procedure: Changing the Picture Geometry Continuously
---|---
7 | For the Picture Window2 object, create a Dynamic Dialog at Properties ➔ Geometry ➔ Window Height. Use the Button to select the tag ➔ U16i_pictu_zoom_00. Use the Button in the Change Trigger dialog to confirm the U16i_pictu_zoom_00 tag as the trigger name and set the standard cycle to Upon Change. Confirm the settings made by clicking on the OK button. In the Data Type field, select the Direct radio-button and exit the Dynamic Dialog by clicking on the Apply button.
8 | For the Picture Window2 object, create a Dynamic Dialog at Properties ➔ Geometry ➔ Window Height. The settings can be made as described above, but the Expression/Formula field must be completed as follow:

<table>
<thead>
<tr>
<th>Expression/Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>U16i_pictu_zoom_00*2</td>
</tr>
</tbody>
</table>

This assigns a value of twice the window width to the window height.
9 | For the pictu_3_chapter_04 picture object, configure a C-Action at Events ➔ Miscellaneous ➔ Open Picture that sets the U16i_pictu_zoom_00 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 Slider Object1. If the Button5 object would then be pressed, the Picture Window2 would be displayed with the dimension of 0x0.

### C-Action for Open Picture

```
#include "apبدфап.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszP)
{
    //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 Button. The necessary picture properties are configured in the properties dialog box.

Implementation in the WinCC Project

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

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

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the \( \text{button} \). 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 $\rightarrow$ Button that will display the picture in a Smart Object $\rightarrow$ Picture Window when pressed with the and two additional Buttons that perform the switching operation and close the panel.

#### Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Binary Switching Operation (Two-Step Control)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create a tag of the Binary Tag type. In the sample, the BINi_pictu_input_00 tag is used. This tag contains the current status of the value.</td>
</tr>
<tr>
<td>2</td>
<td>Configure a picture with two Windows Objects $\rightarrow$ Buttons. In the sample, the pictu_5_window_11 picture is used containing the Button1 and Button2 objects. For Button1, create a direct connection at Events $\rightarrow$ Mouse $\rightarrow$ Press Left. Connect the source Constant $\rightarrow$ 0 with the target Current Window $\rightarrow$ Display. Apply the settings by clicking on the OK button.</td>
</tr>
<tr>
<td>3</td>
<td>At the second Button, Button2 in our sample, configure a C-Action that reverses the status of the binary tag BINi_pictu_input_00.</td>
</tr>
</tbody>
</table>
Step | Procedure: Binary Switching Operation (Two-Step Control)
--- | ---
4 | In another picture, configure a Smart Object ➔ Picture Window; in the sample, this is the Picture Window1. Set the Property ➔ Geometry ➔ Width to 246 and the Property ➔ Geometry ➔ Height to 129. To display the window with a border and make it movable during runtime, set the Property ➔ Miscellaneous ➔ Border to Yes and the Property ➔ Miscellaneous ➔ Moveable to Yes. At Properties ➔ Miscellaneous ➔ Picture Name, select the pictu_3_window_11.pdl picture. Set the Property ➔ Miscellaneous ➔ Display to No.
5 | In the picture, configure a Windows Object ➔ Button. In the sample, this is the Button1 object in the pictu_3_chapter_05.pdl picture. For Button1, create a direct connection at Events ➔ Mouse ➔ Press Left. Connect the source Constant ➔ 1 with the target Object in Picture ➔ Picture Window1 ➔ Display. Apply the settings by clicking on the OK button.

**C-Action at Button2**

```c
#include "apdxap.h"

void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF)
{
  SetTagBit("BIN_pictu_input_00", (SHORT)1);GetTagBit("BIN_pictu_input_00");
}
```

- The status of the BIN_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

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

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

```c
#include "spdefop.h"

void OnButtonDowm(char* lpszPictureName, char* lpszObjectName, char* lpszF
{
    GetTagBit("BINi_pictu_input_02").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 \( \mathcal{B} \). 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 \( \rightarrow \) Button, that will display the picture in the Smart Object \( \rightarrow \) Picture Window when pressed with the \( \mathcal{B} \). In addition, we will use three Windows Objects \( \rightarrow \) Buttons to turn the valve on and off and to close the operator panel. A Smart Object \( \rightarrow \) I/O Field will be used to enter the fill level.

#### Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Automatic Input Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create a tag of the Binary Tag type that contains the current status of the valve. In the sample, the BINi_pictu_input_06 tag is used.</td>
</tr>
<tr>
<td>2</td>
<td>Create two tags of the Unsigned 16-Bit Value type. In this sample, these are the U16i_pictu_input_04 and U16i_pictu_input_05 tags. The first of these two tags contains the set-point value of the container fill level, the second the actual value.</td>
</tr>
<tr>
<td>3</td>
<td>Configure a picture with three Windows Objects ( \rightarrow ) Buttons and a Smart Object ( \rightarrow ) I/O Field. In the sample, the Button1, Button2 and Button3 as well as the I/O Field1 objects are used. As the picture, the pictu_5_window_14.pdl is used.</td>
</tr>
<tr>
<td>4</td>
<td>In the I/O Field1 object's configuration dialog, configure a tag connection to the U16i_pictu_input_04 tag and trigger it upon change.</td>
</tr>
<tr>
<td>5</td>
<td>We assume the container has a maximum fill level of 40 liters. The I/O Field therefore only accepts inputs between 0 and 40. For this, set the Property ( \rightarrow ) Limits ( \rightarrow ) Low Limit Value 0 and the Property ( \rightarrow ) Limits ( \rightarrow ) High Limit Value 40.</td>
</tr>
<tr>
<td>6</td>
<td>For Button1, configure a direct connection at Events ( \rightarrow ) Mouse ( \rightarrow ) Press Left that hides the picture.</td>
</tr>
<tr>
<td>7</td>
<td>For Button2, configure a direct connection at Events ( \rightarrow ) Mouse ( \rightarrow ) Press Left that assigns the value 1 to the BINi_pictu_input_06 tag. For Button3, configure a direct connection that assigns the value 0 to the tag.</td>
</tr>
<tr>
<td>8</td>
<td>In a second picture, configure a Smart Object ( \rightarrow ) Picture Window. In this sample, the Picture Window1 object is used. Adjust the dimension of the Picture Window to match the size of the picture just created. If the Picture Window is to be displayed with a border, the Height and Width of the Picture Window must be 10 pixels greater than those of the picture. At Properties ( \rightarrow ) Miscellaneous ( \rightarrow ) Picture Name, select the pictu_5_window_14.pdl picture.</td>
</tr>
</tbody>
</table>
Step | Procedure: Automatic Input Check
--- | ---
9 | In the same picture, configure a Windows Object Button. In the sample, this is the Button1 object. At Events Mouse Press Left, create a direct connection. Connect the source Constant 1 with the target Object in Picture Picture Window1 Display. Apply the settings by clicking on the OK button.
10 | To display the fill level, the Tank2 object from the library has been used. To simulate the filling process, a C-Action has been created at Properties Geometry Width. At Properties Tag Assignment Fill Level, a tag connection to the U16i_pictu_input_05 tag is configured.
11 | For the second form of displaying the fill level, a Smart Object I/O Field has been used - in the sample, this is the I/O Field1.

C-Action for Simulating the Filling Process

```c
#include "adcsim.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)

{  
    BOOL state;
    SHORT level1,level2:

    //get valve state
    state=GetTagBit("BNW_pictu_input_06");

    if (state==TRUE) {
        level1=GetTagWord("U16i_pictu_input_04");
        level2=GetTagWord("U16i_pictu_input_05");
        level2++;
        if (level2==level1) {
            SetTagBit("BNW_pictu_input_06",FALSE);
            //
            if (level2<=level1) {
                SetTagWord("U16i_pictu_input_05",level2);
                //
            }
        }
    }
    return(60);
}
```

- 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 \(\text{Button}\). 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 \(\rightarrow\) Button, that will display the picture in the Smart Object \(\rightarrow\) Picture Window when pressed with the \(\text{Button}\). Three Smart Objects \(\rightarrow\) I/O Fields are used to enter the fill amounts. In addition, we will use two Windows Objects \(\rightarrow\) Buttons to either apply the settings made in the I/O fields or to cancel them.

#### Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Enhanced Automatic Input Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create two tags of the Binary Tag type that contain the current stati of the valves used to fill the container. In the sample, the BINi_pictu_input_09 and BINi_pictu_input_10 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>Create four tags of the Unsigned 16-Bit Value type. In this sample, these are the U16i_pictu_input_07 , U16i_pictu_input_08, U16i_pictu_input_13 and U16i_pictu_input_14 tags. The first two contain the set-point values for the container fill levels, the last two the actual values.</td>
</tr>
<tr>
<td>3</td>
<td>Create two tags of the Unsigned 16-Bit Value type. In the sample, these are the U16i_pictu_input_11 and U16i_pictu_input_12 tags. These contain the values entered in the I/O Fields.</td>
</tr>
<tr>
<td>4</td>
<td>Configure a picture with two Windows Objects (\rightarrow) Buttons and three Smart Objects (\rightarrow) I/O Fields. In the sample, the Button1 and Button2 as well as the I/O Field1, I/O Field2 and I/O Field3 objects are used. As the picture, pictu_5_window_15.pdl is used.</td>
</tr>
<tr>
<td>5</td>
<td>In the I/O Field1 object’s configuration dialog, configure a tag connection to the U16i_pictu_input_11 tag and trigger it upon change. For I/O Field2, configure a tag connection to the U16i_pictu_input_12 tag.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Enhanced Automatic Input Check</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------</td>
</tr>
</tbody>
</table>
| 6    | For I/O Field3, configure a **Dynamic Dialog at Properties → Output/Input → Output Value**. Enter the settings shown in the figure below. Set the trigger to upon change.  

**Dynamic value ranges**

<table>
<thead>
<tr>
<th>Event name</th>
<th>Tag</th>
<th>Expression/Formula</th>
<th>Apply</th>
<th>Cancel</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>U16i_pictu_input_12+U16i_pictu_input_11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Result Of The Expression/Formula**

<table>
<thead>
<tr>
<th>Valid range</th>
<th>Data Type</th>
<th>Add</th>
<th>Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>U16i_pictu_input_12+U16i_pictu_input_11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Evaluate Status of Tags</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Evaluate Status of Tags</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Evaluate Status of Tags</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7    | For **Button2**, configure a **direct connection at Events → Mouse → Press Left** that hides the picture.  

8    | For **Button1**, create a **C-Action at Events → Mouse → Press Left** that assigns the contents of the input tags **U16i_pictu_input_11 and U16i_pictu_input_12** to the set-point value tags **U16i_pictu_input_07 and U16i_pictu_input_08**. At **Events → Mouse → Mouse Action**, configure a **direct connection** that closes the picture.  

9    | In the same picture, configure two **Standard Objects → Static Texts**. In the sample, the **Static Text5 and Static Text6 objects** are used. These are used to display whether the maximum fill level has been exceeded or not. At the **Static Text5 object**, which contains the error message, set the **Property → Miscellaneous → Display to No.**  

10   | For I/O Field3, create a **C-Action at Events → Property Topics → Output/Input → Output Value** that makes **Button1** operational only if the maximum fill level has not been exceeded and displays the error text if the maximum fill level has been exceeded.
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Enhanced Automatic Input Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>In a second picture, configure a <em>Smart Object</em> → <em>Picture Window</em>. In the sample, the <em>Picture Window2</em> is used. Adjust the dimension of the <em>Picture Window</em> to match the size of the picture just created. At <em>Properties</em> → <em>Miscellaneous</em> → <em>Picture Name</em>, enter <em>pictu_5_window_15.pdl</em>.</td>
</tr>
<tr>
<td>12</td>
<td>In the same picture, configure a <em>Windows Object</em> → <em>Button</em>. In this sample, this is the <em>Button3</em> object. At <em>Events</em> → <em>Mouse</em> → <em>Press Left</em>, create a direct connection. <em>Connect the source Constant</em> → <em>1</em> with the target <em>Object in Picture</em> → <em>Picture Window2</em> → <em>Display</em>. Apply the settings by clicking on the <em>OK</em> button.</td>
</tr>
<tr>
<td>13</td>
<td>To display the fill level, the <em>Tank2</em> object from the library has been used. To simulate the filling process, <em>C-Actions</em> have been created at <em>Properties</em> → <em>Geometry</em> → <em>Width</em> and at <em>Properties</em> → <em>Geometry</em> → <em>Height</em>. At <em>Properties</em> → <em>Tag Assignment</em> → <em>Fill Level</em>, a <em>Dynamic Dialog</em> has been created which returns the sum of the two actual value tags <em>U16i_pictu_input_13</em> and <em>U16i_pictu_input_14</em>.</td>
</tr>
<tr>
<td>14</td>
<td>For the second form of displaying the fill level, a <em>Smart Object</em> → <em>I/O Field</em> has been used - in the sample, this is the <em>I/O Field2</em>.</td>
</tr>
</tbody>
</table>

**C-Action at Button1**

```c
#include "apdefap.h"
void OnButtonDown(char* pszPictureName, char* pszObjectName, char* pszField) {
    // SHORT tmpl, tmp2:
    tmp1=GetTagWord("U16i_pictu_input_11");
    tmp2=GetTagWord("U16i_pictu_input_12");

    if (tmpl)>GetTagWord("U16i_pictu_input_07"){
        SetTagWord("U16i_pictu_input_07",tmpl);
        SetTagBit("BIN_i_pictu_input_07",TRUE);
    }
    if (tmp2)>GetTagWord("U16i_pictu_input_08"){
        SetTagWord("U16i_pictu_input_08",tmp2);
        SetTagBit("BIN_i_pictu_input_08",TRUE);
    }
}
```

- 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

```c
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* s)
{
    int a;
    int 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);
    } else {
        SetOperation(lpszPictureName, "Button1", 0);
        SetVisible(lpszPictureName, "Static Text5", 1);
        SetVisible(lpszPictureName, "Static Text6", 0);
    }
}
```

- 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)

The samples pertaining to this topic are accessed in the Project_CreatePicture project by selecting the Button displayed above using the \(\square\). 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 \(\rightarrow\) Buttons, that will display the picture in the Smart Object \(\rightarrow\) Picture Window when pressed with the \(\square\). Two Windows Objects \(\rightarrow\) 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 \(\rightarrow\) Static Texts. The picture is anchored via a Smart Object \(\rightarrow\) Status Display.

Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Multiple Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create tags of the Binary Tag type that display the current stati of the valves. The number of tags required depends on the number of valves. In the sample, the BINi_pictu_multi_01, BINi_pictu_multi_02, BINi_pictu_multi_03 and BINi_pictu_multi_04 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>Create a tag of the Text Tag 16-Bit Character Set type. In the sample, this is the T16x_pictu_input_15 tag. This tag will be used as an address tag.</td>
</tr>
<tr>
<td>3</td>
<td>Create a tag of the Binary Tag type. In the sample, this is the BINi_pictu_multi_00 tag. The content of this tag contains information whether the window has been anchored.</td>
</tr>
<tr>
<td>4</td>
<td>Configure a picture with three Windows Objects (\rightarrow) Buttons. In the sample, the Button1, Button2 and Button3 objects are used. As the picture, pictu_5_window_16.pdl is used.</td>
</tr>
<tr>
<td>5</td>
<td>For Button1, create a C-Action at Events (\rightarrow) Mouse (\rightarrow) Press left that sets the position of the picture outside the visible area, closes the picture and cancels the anchoring of the picture.</td>
</tr>
<tr>
<td>6</td>
<td>For Button2, create a direct connection at Events (\rightarrow) Mouse (\rightarrow) Press Left. Connect the source Constant (\rightarrow) 1 with the target Variable (\rightarrow) T16x_pictu_input_15. 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 Button2 with the source Constant (\rightarrow) 0.</td>
</tr>
</tbody>
</table>
7 **Configure a Smart Object ➔ Status Display.** In the sample, the Status Display1 is used. In the following configuration dialog, select the BINi_pictu_multi_00 tag and set the trigger to upon change. Use the Button Add to add another status. For the 0 status, select the put_up.gif picture and for the 1 status, the put_down.gif picture.

8 For the Status Display1, create a C-Action at Events ➔ Mouse ➔ Press Left that negates the status of the BINi_pictu_multi_00 tag.

9 For the title, configure a Standard Object ➔ Static Text. In this sample, the Static Text1 object is used. At Properties ➔ Font ➔ Text, create a C-Action that reads the current valve number from the T16x_pictu_input_15 address tag and returns an appropriate text as the return value.

10 Configure another Standard Object ➔ Static Text to display the valve status. In the sample, the Static Text2 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.

11 In a second picture, configure a Smart Object ➔ Picture Window. In this sample, the Picture Window1 object is used. Adjust the dimension of the Picture Window 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 pictu_5_window_16.pdl picture.

12 In the same picture, configure a Windows Object ➔ Button for each valve; in the sample the Button1, Button2, Button3 and Button4 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.
C-Action for the Close Button (Button1)

```c
#include "apdsian.h"

void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
    SetLeft("picture_3_chapter_05b", "Picture Window1", -1000);
    SetVisible("picture_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

```c
#include "apdsian.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

```c
#include "opdefsp.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszPName){
    int x, y;
    char name[20];
    int number;
    int ch = n;
    char *pdest;
    // check if object name contains character n
    pdest = strchr(lpszObjectName, ch);
    if (pdest == NULL)(printf("ObjectNameError"));
    // read object number
    else{
        number = atoi(strchr(lpszObjectName, n)+1);
        sprintf(name, "%s\N\_multi\_%d\D", number);
        // generate tag name
        SetTagChar("\T\N\_picture\_input\_1", name);
    }
    setVisible(lpszPictureName, "Picture Window1", 1);
    if (GetLeft(lpszPictureName, "Picture Window1") == FALSE){
        // set object position
        x = GetLeft(lpszPictureName, lpszObjectName);
        y = GetTop(lpszPictureName, lpszObjectName);
        // set position of picture window
        SetLeft(lpszPictureName, "Picture Window1", x-1000);
        SetTop(lpszPictureName, "Picture Window1", y-1000);
        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

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the \( \mathcal{V} \). 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 \( \rightarrow \) Slider Object that will change the value of a tag. The text display is implemented via a Standard Object \( \rightarrow \) Static Text.

Implementation in the WinCC Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Color Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create a tag of the <strong>Signed 32-Bit Value</strong> type. In the sample, this is the <strong>S32i_pictu_dyn_00</strong> tag.</td>
</tr>
<tr>
<td>2</td>
<td>Configure a <strong>Windows Object ( \rightarrow ) Slider Object</strong>. In this sample, the <strong>Slider Object1</strong> is used. In the <strong>configuration dialog</strong>, set the <strong>Maximum Value</strong> to 1000 and the <strong>Minimum Value</strong> to 0. At <strong>Events ( \rightarrow ) Property Topics ( \rightarrow ) Miscellaneous ( \rightarrow ) Process Driver Connection</strong>, create a <strong>direct connection</strong> to the <strong>S32i_pictu_dyn_00</strong> tag.</td>
</tr>
<tr>
<td>3</td>
<td>Configure a <strong>Standard Object ( \rightarrow ) Static Text</strong>. In this sample, the <strong>Static Text5</strong> object is used. At <strong>Properties ( \rightarrow ) Font ( \rightarrow ) Text</strong>, create a <strong>C-Action</strong> that outputs a text with the corresponding tag value. This C-Action is triggered upon the change of the tag.</td>
</tr>
<tr>
<td>4</td>
<td>At <strong>Properties ( \rightarrow ) Colors ( \rightarrow ) Font Color</strong>, create a <strong>Dynamic Dialog</strong>. In the <strong>Expression/Formula</strong> field, set the <strong>S32i_pictu_dyn_00</strong> tag and trigger it upon change. Select the <strong>Data Type Analog</strong> and add 4 value ranges via the <strong>Add</strong> button. Set the value ranges as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid range</th>
<th>Up to</th>
<th>Font color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value Range1</td>
<td>100</td>
<td>Red</td>
</tr>
<tr>
<td>Value Range2</td>
<td>300</td>
<td>Blue</td>
</tr>
<tr>
<td>Value Range3</td>
<td>600</td>
<td>Green</td>
</tr>
<tr>
<td>Value Range4</td>
<td>1000</td>
<td>Orange</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>Other</td>
</tr>
</tbody>
</table>
Step | Procedure: Color Change
---|---
5 | At Properties ➔ Flashing ➔ Flashing Background Active, create a Dynamic Dialog. In the Expression/Formula field, set the $S32i\_pictu\_dyn\_00$ tag and trigger it upon change. Select the Data Type Analog and add a value range via the Add button. Set the value range as follows:

### Dynamic value ranges

- **Tag**: Create a Dynamic Dialog. In the **Expression/Formula** field, set the **$S32i\_pictu\_dyn\_00$** tag and trigger it upon change. Select the **Data Type Analog** and add a value range via the **Add** button. Set the value range as follows:

  - **Value Range**: 300
    - **Valid range**: No
    - **Up to**: Yes

### C-Action at Static Text

```c
#include "_end_of_h"

char* _main(char* lipszPictureName, char* lipszObjectName, char* lipszPropro
{
    char text[100];
    DWORD temp;

    // get tag value
    temp = GetTagDWord("$S32i\_pictu\_dyn\_00$"));

    // generate text
    switch (GetLanguage())
    {
        case 0x40:
            sprintf(text, "Die Kesseltemperatur beträgt\n\n%d Grad", temp);
            return text;
        case 0x409:
            sprintf(text, "Container Temperature is\n\n%d degree", temp);
            return text;
        case 0x40C:
            sprintf(text, "La température de chaudière est\n\nde %d degré", temp);
            return text;
        default:
            sprintf(text, "Container Temperature is\n\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

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

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 $\rightarrow$ Picture Window, whose position is controlled by a tag. A Windows Object $\rightarrow$ Slider Object is used to change the values of the tag.

Implementation in the WinCC Project

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Displaying and Hiding Objects using a Bit Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create a tag of the Unsigned 8-Bit Value type. In this sample, the U08_pictu_dyn_02 tag is used.</td>
</tr>
<tr>
<td>2</td>
<td>Configure a Windows Object → Check-Box; in this sample the Check-Box1 object is used. At Properties → Geometry → Number of Boxes, set the number of objects to be switched; in the sample, this is 7. At Properties → Font → Text, enter the name of the object that is to be switched by the corresponding bit for each index value. At → Events → Property Topics → Output/Input → Selected Boxes → Change, create a direct connection with the source Property → Check-Box1 → Selected Boxes and the target Variable U08i_pictu_dyn_02.</td>
</tr>
<tr>
<td>3</td>
<td>Configure several Standard Objects → Polygon. In this sample, the Polygon1 to Polygon7 objects are used.</td>
</tr>
<tr>
<td>4</td>
<td>For the Polygon1 object, create a Dynamic Dialog at Properties → Miscellaneous → Display. In the Expression/Formula field, set the U08i_pictu_dyn_02 tag and trigger it upon change. Select the Data Type Bit. Use the button to open the bit selection dialog and select the first bit.</td>
</tr>
<tr>
<td>5</td>
<td>Proceed in the same manner for the remaining Polygon objects, but change the bit number of each one.</td>
</tr>
</tbody>
</table>
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途．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**

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

```c
#include <stdio.h>
long main(char* lpszPropertyName, char* lpszObjectName, char* lpszPropertyN
{
    static int a = 90;

    // forward
    if (GetTagBit("BINi_pictu_dyn_07")&& (a<652)) {
        SetTagBit("BINi_pictu_dyn_05", (SHORT)!GetTagBit("BINi_pictu_dyn_05").
    }

    // rewind
    if (GetTagBit("BINi_pictu_dyn_06")&& (a>0)) {
        a--20;
        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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creating Animation of Movement with a Wizard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create two tags of the Unsigned 32-Bit Value type. In this sample, the S32i_pictu_dyn_10 and S32i_pictu_dyn_11 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>Configure two Windows Objects ➔ Slider Objects; in this sample, Slider Object1 and Slider Object2 are used. For Slider Object1, create a direct connection. Connect the source Properties ➔ Slider Object1 ➔ Process Driver Connection with the Variable S32i_pictu_dyn_10. In the same manner, create a direct connection for Slider Object2 to the Variable S32i_pictu_dyn_11.</td>
</tr>
<tr>
<td>3</td>
<td>In the configuration dialogs of the Slider Objects, set the Maximum Value to 255.</td>
</tr>
<tr>
<td>4</td>
<td>Configure a Standard Object ➔ Circle. In this sample, the Circle1 object is used. While the object is highlighted, select the Standard Dynamics tab and then the Move Object entry from the Dynamic Wizard via a ( \sqrt{D} ). As the trigger, select Tag. On the Set Options 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 Finish to complete the Wizard.</td>
</tr>
<tr>
<td>5</td>
<td>In the C-Actions generated by the Dynamic Wizard, set the trigger to upon change for the corresponding tag used at Properties ➔ Geometry ➔ Position X and at Properties ➔ Geometry ➔ Position Y.</td>
</tr>
</tbody>
</table>

C-Action generated by the Wizard at Position X

```c
#include "apdswoc.h"

long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    long i, j, k;
    i=GetIntWord("S32i_pictu_dyn_10");
    j=(i-0)*100/(255-0);
    k=min(((j*690-490)/100)+490,690);
    return round(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

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

```c
#include "amstap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    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

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

```c
#include "apdpclap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    static int a = 0, b = 0;
    if (GetTagBit('BIHi_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

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

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

C-Action for the German Button

```c
#include "apptap.h"

void OnButtonDown(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

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

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

The samples pertaining to this topic are accessed in the *Project_CreatePicture* project by selecting the *Button* displayed above using the \*\j. 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 resetable 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuring the Cursor Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Control Center, make the settings for the Cursor Control. On the Computer entry and then select Properties from the pop-up menu. In the following Computer List Properties dialog, click on the Button Properties. Select the Graphics Runtime tab.</td>
</tr>
</tbody>
</table>

2 Make the hotkey settings as follows. For the Window on Top command, no hotkey is configured, since in the samples the operating focus is set via C-Actions.

To switch between the Tab Order/Alpha Cursor, set SHIFT+A and to turn the Runtime Cursors On/Off, set SHIFT+R.
### Implementation in the WinCC Project

#### Procedure: Configuring the Cursor Control

3. In the **Cursor Control: Keys** 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.

#### Procedure: Configuring the Hotkey Operation

1. In Tag Management, create three tags of the **Unsigned 16-Bit Value** type that will contain the font properties set. In this sample, the `U16i_pictu_cursor_00` to `U16i_pictu_cursor_02` tags are used.

2. In the `pictu_3_chapter_08.pdl` picture, configure four objects of the **Windows Object** → **Button** type. In this sample, these are the `Button1`, `Button2`, `Button3` and `Button4` objects. They used to display the dialogs and to reset the settings made.

   In addition, configure a **Standard Object** → **Static Text**, whose font properties are set via the dialogs. In this sample, the `Static Text1` object is used.

3. Configure another picture which serves as the dialog for setting the color. In this sample, this is the `pictu_5_window_23.pdl` picture.

   In this picture, configure a **Windows Object** → **Option Group**. In this sample, this is the `Option Group1` object. Set the Property → **Geometry** → **Number of Boxes** to 4. This enables the selection of four different colors.

   At **Properties** → **Output/Input** → **Selected Boxes**, create a tag connection to the `U16i_pictu_cursor_00` tag.

   At **Properties** → **Geometry** → **Position X**, create a C-Action that sets the focus to object just created. This C-Action is triggered every 1 h. If the **Option Group** 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 **Windows Objects** → **Buttons**. In the sample, these are the `Button1` and `Button2` objects.

   `Button1` is used as the **OK Button**. At **Events** → **Mouse** → **Mouse Action**, create a C-Action that changes the color of the text depending on the value of the `U16i_pictu_cursor_00` tag and then hides this dialog.

   `Button2` is used as the **Cancel Button**. At **Events** → **Mouse** → **Mouse Action**, create a direct connection that makes the window invisible.
### Procedure: Configuring the Hotkey Operation

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Make the settings for the keyboard operation. Set the Tab order. This is done via the <em>Edit</em> → <em>TAB Sequence</em> → <em>Tab Order</em> menus.</td>
</tr>
</tbody>
</table>

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.

The order is set as follows:

#### Set Color

- Black
- Red
- Green
- Blue

[Image of a window with options for setting colors]
### Procedure: Configuring the Hotkey Operation

The selection in the *Option Group* is performed via the *cursor keys*. The selection of a color is performed via the *spacebar*. The *TAB key* is used to switch among the operating elements. The buttons are operated via the *space bar*.

In addition, hotkeys are assigned to both *Buttons*. Via *Properties* → *Miscellaneous* → *Hot Key*, the dialog for the configuration of a hotkey is opened. For the *OK Button*, the *ENTER key* is set, for the *Cancel Button*, the *ESC key* is set.

#### Function Keys:

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F9</th>
<th>F10</th>
<th>F11</th>
<th>F12</th>
</tr>
</thead>
</table>

**Trigger with**: EINGABE

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

---

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>6</strong></td>
<td>In the <em>pictu_3_chapter_08.pdl</em> picture, configure a <em>Smart Object</em> → <em>Picture Window</em>, in which the picture just configured is displayed. In this sample, this is the <em>Picture Window1</em> object. At <em>Properties</em> → <em>Miscellaneous</em> → <em>Picture Name</em>, set the <em>pictu_3_window_23.pdl</em> picture. Set the <em>Property</em> → <em>Miscellaneous</em> → <em>Display</em> to <em>No</em>.</td>
</tr>
<tr>
<td><strong>7</strong></td>
<td>For <em>Button1</em>, create a <em>C-Action</em> that queries the currently set color of the text and writes to the <em>U16i_pictu_cursor_00</em> tag depending on the result. This is done to set the selection in the <em>Option Group</em> of the dialog to the currently set value. In addition, the <em>Picture Window1</em> object is displayed. For <em>Button1</em>, create a <em>C-Action</em> at <em>Properties</em> → <em>Geometry</em> → <em>Position X</em> that sets the operating focus to this object. This <em>C-Action</em> is triggered every 1 h, however the focus is only set at the first time. <strong>Set Color</strong> → <strong>F9</strong></td>
</tr>
</tbody>
</table>

In addition, a hotkey is assigned to this *Button*. In the sample, this is the *F9* function key.
### Step Procedure: Configuring the Hotkey Operation

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
</tr>
</thead>
</table>
| 8    | Configure a picture that will serve as the dialog for setting various font properties. In the sample, this is the `pictu_5_window_24.pdl` picture.  

In this picture, configure a **Windows Object** ➔ **Check-Box**. In this sample, the **Check-Box1** object is used. Set the **Property** ➔ **Geometry** ➔ **Number of Boxes** to 4. The selection of the **Bold**, **Italic**, **Underline** and **Border** properties is to be made possible.  

At **Properties** ➔ **Output/Input** ➔ **Selected Boxes**, create a tag connection to the `U16i_pictu_cursor_01` tag.  

At **Properties** ➔ **Geometry** ➔ **Position X**, create a **C-Action** that sets the focus to this object. The selection frame can be hidden using the same procedure as described for the **Option Group**.  

Just like for the `pictu_5_window_23.pdl` picture, configure two **Windows Objects** ➔ **Buttons**. If the **OK Button** is activated, the `U16i_pictu_cursor_01` tag is read and the corresponding settings are applied to the text.  

The settings pertaining to the keyboard operation are made in the same manner as for the `pictu_5_window_23.pdl` picture. |
| 9    | In the `pictu_3_chapter_08.pdl` picture, configure another **Smart Object** ➔ **Picture Window**, in which the picture just configured is displayed. In this sample, this is the **Picture Window2** object. At **Properties** ➔ **Miscellaneous** ➔ **Picture Name**, set the `pictu_3_window_24.pdl` picture. Set the **Property** ➔ **Miscellaneous** ➔ **Display** to **No**. |
| 10   | For **Button2**, create a **C-Action** that queries the currently set font properties to be changed and writes them to the `U16i_pictu_cursor_01` tag depending on the result. This is done to set the selections in the **Check-Box** of the dialog to the currently set values. In addition, the **Picture Window2** object is displayed.  
Furthermore, a hotkey is assigned to this **Button**. In the sample, this is the **F10** function key. |

**Format** F10
### Step | Procedure: Configuring the Hotkey Operation
--- | ---
11 | Configure another picture that will serve as the dialog for setting the font size. In this sample, this is the *pictu_5_window_25.pdl* picture.

   In this picture, configure a **Smart Object → I/O Field**. In this sample, the **I/O Field1** object is used.

   At Properties → Output/Input → Output Value, create a tag connection to the **U16i_pictu_cursor_02** tag.

   At Properties → Geometry → Position X, create a C-Action that sets the focus to this object. The selection frame is hidden by positioning a **Graphic Object** on top of the **I/O Field**. In this sample, the **Graphic Object1** is used. The bitmap displayed by the **Graphic Object** has a certain color in the area where the **I/O Field** is to be displayed. This color in the **Graphic Object** is set at Properties → Picture → Picture Transparent Color. Additionally, the Property → Picture → Picture Transparent Color On is set to Yes. The following displays the bitmap used.

![Bitmap](image1)

12 | Just like for the *pictu_5_window_23.pdl* picture, configure two **Windows Objects → Buttons**. If the **OK Button** or the **Cancel Button** is pressed, the dialog is closed.

   Both **Buttons**, however, are excluded from the TAB order. This is done via the *Edit → TAB Sequence → Tab Order → Sequence* menus. An object can be removed from the TAB order by pressing and holding the **CTRL+SHIFT** key combination and selecting it with the ✂. Instead of the number, a * is displayed in the white rectangle.

   ![Button Symbol](image2)

   The operation of the **Buttons** is to be performed exclusively via the hotkeys **ENTER** and **ESC**. If the **ENTER** key is pressed, the value entered in the **I/O Field** is transferred to the **U16i_pictu_cursor_02** tag.

   For the **Static Text1** object in the *pictu_3_chapter_08.pdl* picture, create a tag connection at Properties → Font → Font Size to the **U16i_pictu_cursor_02** tag.

13 | In the *pictu_3_chapter_08.pdl* picture, configure another **Smart Object → Picture Window**, in which the picture just configured is displayed. In this sample, this is the **Picture Window3** object. At Properties → Miscellaneous → Picture Name, set the **pictu_3_window_25.pdl** picture. Set the Property → Miscellaneous → Display to No.
### C-Action for Setting the Focus

```c
#include 'opcettop.h'

long _main(char* lpszPropertyName, char* lpszObjectName, char* lpszPropName)
{
    static BOOL bFirst = FALSE;
    //set focus in first run
    if (bFirst==FALSE)
        Set_Focus(lpszPropertyName,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

```c
#include "spdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
    long int vValue;
    //get text color
    vValue=GetForeColor(lpszPictureName, "Static Text1");
    //set tag switch contains text color
    switch (vValue)
    {
        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 dialogue 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

```c
#include "spdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
    //load DLL
    #pragma code ("pdlrtps1.dll")
    #include <pdlrtps1.h>
    #pragma code ()
    PDLRTRSetCursorKeys(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 PDLRTRSetCursorKeys. 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 checkboxes, however, the cursor keys are used by default.
3.9.2 Cursor Keyboard (example 02)

<table>
<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>

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 `CTRL`. 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**

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, two tags of the Unsigned 16-Bit Value type are created that will store the cursor behavior set. In this sample, the <code>U16i_pictu_cursor_04</code> and <code>U16i_pictu_cursor_05</code> tags are used. Additionally, a tag of the Text Tag 16-Bit Character Set type is created to which the entered text is transferred. In the sample, the <code>T16i_pictu_cursor_00</code> tag is used.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Implementation in the Graphics Designer</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Open the Library via the button on the toolbar. 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:</td>
</tr>
</tbody>
</table>

![Keyboard Layout](image)

3 For the **ENTER** Button, create a C-Action at **Events → Mouse → Mouse** Action that writes the text entered into a text tag. The name of this tag is ConnectedVarChar. Change this name to T16i_pictu_cursor_00. In this sample, the content of the T16i_pictu_cursor_00 tag is displayed in a static text having the title of the picture. This is carried out via a tag connection to this tag.

4 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.

The settings for the cursor keys of the Cursor Control are made in a **C-Action** at **Events → Miscellaneous → Open Picture**.

5 The cursor behavior can be set via a dialog box.

In normal cases, this is already performed in the **Control Center**. **R** on the **Computer** entry and then select **Properties** from the pop-up menu. In the following **Computer List Properties** dialog, click on the **Button Properties**. Select the **Graphics Runtime** tab. In the **Cursor Control: Keys** field, three settings can be made.
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
</table>
| 6    | Create a new picture, which serves as the dialog window. In this sample, this is the `pictu_5_window_26.pdl` picture.  
In this picture, configure three **Smart Objects** → **Status Displays**. In the sample, these are the `Status Display1`, `Status Display2` and `Status Display3` objects. Via the configuration dialog, set bitmaps for each `Status Display` that display the pressed status of a button and the not pressed status of a button. The `Status 1` represents the pressed button and the `Status 0` the not pressed button.  
**At Properties** → **State** → **Current Status**, create a **Dynamic Dialog** each that controls the current status depending on the `U16i_pictu_cursor_05` tag. This tag contains the temporarily made settings of the cursor behavior.  
**At Events** → **Keyboard** → **Press**, create a **C-Action** that writes a value to the `U16i_pictu_cursor_05` tag representing a certain selection. These values are: `0...Normal`, `1...Normal without Rollback`, `10...Extended`. |
C-Action at the Press Keyboard Event

```c
#include "admefap.h"
void OnKeyDown(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
  if (nChar==VK_SPACE)
    SetTagWord("016i_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

```c
#include "admefap.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("016i_pictu_cursor_04",U16i_pictu_cursor_04);
  //set cursor mode
  PDLRTSetCursorKeys(VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, U16i_pictu_cursor_04);
  //set focus to A-button
  SetFocus("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 \( \sqrt{1} \). 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 \( \rightarrow \) Buttons are used to which hotkeys are assigned. Values are to be entered in Smart Objects \( \rightarrow \) I/O Fields and valves turned on and off.

Implementation in the Graphics Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create six tags of the Signed 16-Bit Value type that are used to make the entries and to store them afterwards. In the sample, these are the S16i_pictu_cursor_00 to S16i_pictu_cursor_05 tags.</td>
</tr>
</tbody>
</table>
| 2    | In a picture, configure three Smart Objects \( \rightarrow \) I/O Fields in which the fill level set-point values are to be entered. In this sample, these are the I/O Field1, I/O Field2 and I/O Field3 objects.  
For I/O Field1, create a tag connection in the configuration dialog to the S16i_pictu_cursor_00 tag and set the High Limit Value to 4999 and the Low Limit Value to 0.  
Proceed in the same manner for the remaining I/O Field, but set the S16i_pictu_cursor_01 or S16i_pictu_cursor_02 tags respectively. For I/O Field3, set the Upper Limit Value to 9999. |
## Step Procedure: Implementation in the Graphics Designer

### 3
Configure three **Windows Objects** → **Buttons** that are used to apply the values entered in the I/O Fields. In this sample, these are the **ButtonF6**, **ButtonF7** and **ButtonF8** objects.

For **ButtonF6**, create a **C-Action at Events** → **Mouse** → **Mouse Action** that writes the value entered in the **S16i_pictu_cursor_00** tag to the **S16i_pictu_cursor_03** tag. Via **Properties** → **Miscellaneous** → **Hotkey**, set the **F6** hotkey for the **Button**.

Proceed in the same manner for the remaining **Buttons**.

For **ButtonF6**, create a **C-Action at Properties** → **Geometry** → **Width** that sets the focus to this button.

### 4
Configure three **Standard Objects** → **Rectangles** that represent the values entered. In this sample, these are the **Rectangle9**, **Rectangle10** and **Rectangle11** objects.

Set the **Property** → **Filling** → **Dynamic Filling** to **Yes**. At **Properties** → **Filling** → **Fill Level**, create a **Dynamic Dialog** for each rectangle to convert the tag value to a fill level.

For the graphical display of the container, several grouped **Standard Objects** are used.

### 5
Configure four additional **Windows Objects** → **Buttons**. These buttons are used to turn valves on and off. In this sample, these are the **ButtonF9**, **ButtonF10**, **ButtonF11** and **ButtonF12** objects.

At **Events** → **Mouse** → **Mouse Action**, configure a **C-Action** for each button that inverts a binary tag representing a valve status. Each **Button** is assigned a hotkey.

### 6
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 **Displaying and Hiding Information**, in the sample Displaying and Hiding Objects (example 01).
Step | Procedure: Implementation in the Graphics Designer
--- | ---
7 | Except for the Buttons to which hotkeys have been assigned, all objects are removed from the TAB order.
   Via the Edit → TAB Sequence → Alpha Cursor → Sequence menus, the order in which the I/O Fields are selected with the TAB key is defined.

8 | In the Control Center, define a key combination that switches between the TAB Order and the Alpha Cursor. Click on the Computer entry and then select Properties from the pop-up menu. In the following Computer List Properties dialog, click on the Button Properties. Select the Graphics Runtime tab.
   To switch between the TAB Order/Alpha Cursor, the key combination SHIFT+A is set. Additionally, the key combination SHIFT+R is set to turn the runtime cursor on and off.

**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

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.pdf` 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Displaying and Hiding Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create three tags of the Binary Type, which control the visibility of the various object groups. In this sample, the BINi_pictu_info_12, BINi_pictu_info_13, and BINi_pictu_info_14 tags are used.</td>
</tr>
<tr>
<td>2</td>
<td>In Tag Management, create additional tags of the Binary Tag type that contain the current status of the valves. The number of tags required depends on the number of valves. In the sample, the BINi_pictu_info_1 to BINi_pictu_pictu_11 tags are used for a total of 11 valves.</td>
</tr>
<tr>
<td>3</td>
<td>To display an open valve, a Standard Object Polygon is configured, which has the shape of a valve. At Properties Colors Background Color, set the color Dark Green.</td>
</tr>
<tr>
<td>4</td>
<td>To display a closed valve, a Standard Object Polyline is configured, which has the shape of a valve.</td>
</tr>
<tr>
<td>5</td>
<td>Configure two identical Standard Objects Rectangles and set the background color of the picture at their Properties Color Background Color. The rectangles should be slightly larger than the valves, so that they can hide them.</td>
</tr>
<tr>
<td>6</td>
<td>Position a rectangle and an open valve one on top of each other and set the open valve to the foreground by clicking on the Button Group both objects via the Edit Group Object menus. For the Group Object generated, configure a tag connection to the BINi_pictu_info_1 tag at Properties Miscellaneous Display.</td>
</tr>
</tbody>
</table>
### Step 7
**Procedure: Displaying and Hiding Objects**
Position the closed valve on top of the second rectangle and set to the foreground. Then position the group object 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 group object can be copied. Only the tag connection must be adapted.

### Step 8
For each valve, configure a Windows Object ➔ Button and create a C-Action at Events ➔ Mouse ➔ Press Left that negates the corresponding tag value.

### Step 9
For each valve, configure a Standard Object ➔ Static Text that contains the name of the valve.

### Step 10
Configure several containers whose fill levels are displayed via Smart Objects ➔ I/O Fields.

### Step 11
Configure three Windows Objects ➔ Buttons. In this sample, the Button12, Button13 and Button14 objects are used. For Button12, create a C-Action at Events ➔ Mouse ➔ Press Left that negates the value of the BINi_pictu_info_12 tag. For the remaining Buttons, create C-Actions in the same manner for the BINi_pictu_info_13 and BINi_pictu_info_14 tags.

### Step 12
For all objects that are displayed or hidden via Button12, create a tag connection to the BINi_pictu_info_12 tag. Do exactly the same for the other objects. In this sample, Button12 makes visible the I/O Fields, Button13 the Static Texts and Button14 the Buttons.

### 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

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

C-Action for Reading the Time

```c
#include <sys/time.h>
#include <sys/types.h>

char* _asm(char* lpszObjectName, char* lpszPropertyName)
{
    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

```c
#include "apds9912.h"
char* _main(char* lpstrPictureName, char* lpstrObjectName, char* lpstrProperty
{
    time_t timer;
    struct tm *ptm;
    char *p;

    ptm=localtime(&timer);
    p=SysMalloc(9);
    sprintf(p,"%02d:%02d:%02d",ptm->tm_hour,ptm->tm_min,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 \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

In runtime, the samples pertaining to this topic are accessed by selecting the button displayed above using the $\text{ }$. 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.

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

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

4 The start page is exited by clicking on the Next button. On the next page, set as the Archive Type the process value archive option. Enter an Archive Name. In this sample, the archive has been named ZK_ProcessValueArchive_00. Continue to the next page by clicking on Next.

5 On the third page of the Wizard, the tags to be archived are defined. This is done via the Select button. In this sample, the G64_ex_tlg_01, G64_ex_tlg_02 and G64_ex_tlg_03 tags are used. Close this page of the Wizard by clicking on Finish.
### Procedure: Creating a Process Value Archive

**Step 6**

On the right side of the window, the entry of the newly created archive will be displayed.

Via a `R` on this entry and then selecting *Properties*, the properties dialog of this archive is opened.

- **Archive Name**: `ProcessValueArchive_00`
- **Server Name**: `ZIP-WS5`
- **Archiving at System Start**: Enabled
- **Archive Type**: Process Value Archive
- **Authorization Level**: No Access Protection

**Step 7**

In the *General Information* tab, the basic archive parameters can be set. The *Archive Name* and the *Archive Type* have already been specified in the *Archive Wizard*. The *Archive Type* cannot be changed anymore.

The *Archiving at System Start* is enabled.

This initiates the archiving directly after the system start. It is not necessary to enable the archive via a separate function. In the *Authorization Level* field, *No Access Protection* 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.

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.
Step 8

Procedure: Creating a Process Value Archive

In the Archive Parameters tab, additional properties of the archive are set. As the archive size, 1000 data records is set. As the Storage Location, select On the Hard Disk. As the Archive Type, select Short-Term Archive. As the Action for Exporting the Short-Term Archive, a function can be set that is executed automatically if the short-term archive is full. For this sample, no action is specified.

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.

Close the properties dialog of the archive by clicking on OK.

Step 9

Specify the properties of the individual archive tags.

For this purpose, on the lower table window and select Properties from the pop-up menu to open the properties dialog of an archive tag.
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creating a Process Value Archive</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>In the Archive Tag tab, the basic tag properties can be set. The corresponding process tag has already been specified in the Archive Wizard. As the Name of the Archive Tag, a name can be assigned; in this sample, however, the name of the corresponding process tag is used. In the Supplying Tags field, the System radio-button is selected. In the Archiving at System Start field, the Enabled radio-button is selected. As the Archiving Type, Cyclic-Continuous 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. The archived value is not written to a tag.</td>
</tr>
</tbody>
</table>

![Properties of process tag](image)
Step 11

Procedure: Creating a Process Value Archive

In the Parameters tab, additional settings are made.

In the Cycle field, set as the Acquisition 500 ms and as the Archiving 1*500 ms.

In the Processing field, select the Actual Value radio-button.

A Unit is not specified. In the case of an error, the last value is to be saved. The Archiving upon Change check-box is not selected.
12 In the Display tab, the acceptance area of the tag into the archive is specified. In this sample, the No Display Limitation radio-button is selected.

In the Display dialog box, you configure the display range or transfer range into the archive of the tags.
### Procedure: Creating a Process Value Archive

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>In the <em>Events</em> tab, no action for changing the archiving cycle is entered for this sample in the <em>Dynamic</em> field. Close the properties dialog of the archive tag by clicking on <em>OK</em>.</td>
</tr>
</tbody>
</table>

![Properties of process tag](image)

14 The properties of the two other archive tags must also be specified. For this purpose, steps 9 to 13 must be performed.

**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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuring the Trend Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a new picture in the <em>Graphics Designer</em>. In this sample, this is the <em>ex_3_chapter_01.pdl</em> picture.</td>
</tr>
<tr>
<td>2</td>
<td>Configuration of the Control used for displaying the trend profiles. This is the <em>WinCC Online Trend Control</em>. It is selected from the <em>Object Palette’s Control</em> selection menu and then placed in the picture.</td>
</tr>
</tbody>
</table>

![Object Palette Diagram]
Step | Procedure: Configuring the Trend Display
--- | ---
3 | After placing the Control in the picture, its configuration dialog will be opened automatically.

In the General Information tab, you can specify the Control's title and how it is labeled. In this sample, the Display check-box is selected and as the Window Title the name of the previously created archive ZK_ProcessValueArchive_00 is entered.

In the Open Picture 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 Load Data from Archive check-box. If this is not done, the Control will only display values that have been archived after the picture has been opened.

In the Supplying Data field, you can select whether the profiles of Archive Tags or Online Tags are to be displayed. If Online Tags is selected, the trend profiles of tags that are not archived can also be displayed. For this sample, Archive Tags is set.

Via the Color button, the Background Color 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 Object Properties dialog of the Control1 object.

In the Display field, this sample specifies the display of the Toolbar and Status Bar. As the Ticker Direction, From the Right is selected. In addition, a Shared X Axis and Shared Y Axis is used. The window size is not to be changed.
### 4. Configuring the Trend Display

In the *Trends* 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 `Trend_G64_ex_tlg_01`. The Display Type *Connect Dots Linearly* is kept. Via the *Selection* button, the archive tag to be displayed can be assigned to the trend.

#### Properties of WinCC Online Trend Control

<table>
<thead>
<tr>
<th>Trends</th>
<th>Name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Trend_G64_ex_tlg_01</td>
<td>Trend_G64_ex_tlg_01</td>
</tr>
<tr>
<td>✓ Trend_G64_ex_tlg_02</td>
<td></td>
</tr>
<tr>
<td>✓ Trend_G64_ex_tlg_03</td>
<td></td>
</tr>
</tbody>
</table>

Display:
- Visible
- Color...

Selection of Archives/Tags:
- ZK_ProcessValueArchive_00\G64_ex_

Display type:
- Connect dots linearly

### 5. Archive/Tag Selection

The *Archive/Tag Selection* dialog is displayed. In the left window, select the desired `ZK_ProcessValueArchive_00` archive. In the right window, select the desired `G64_ex_tlg_01` archive tag available in this archive.

Exit the dialog box by clicking on the *OK* button.
### Procedure: Configuring the Trend Display

**Step 6**
Create two additional trends to display the remaining Archive Tags.
A new trend is added in the *Trends* tab by clicking on the + button.
Their properties are set as described in steps 4 to 5. However, the archive tags `G64_ex_tlg_02` and `G64_ex_tlg_03` are used.
The properties dialog of the Control can be closed by clicking on OK.

![Properties of WinCC Online Trend Control](image)

**Step 7**
Setting the background color of the trend window. For this purpose, select R and then select Properties from the pop-up menu to open the Object Properties dialog of the Control1 object.

![Setting the background color](image)
## Procedure: Configuring the Trend Display

In this sample the *BackColor* is matched to the color scheme used in the project. You can also make all the settings of the WinCC User Archives Table Control Properties dialog in here. For some settings, however, this is not useful.
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuring the Trend Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Activation of Tag Logging Runtime.</td>
</tr>
<tr>
<td></td>
<td>For this, open the Computer entry in the</td>
</tr>
<tr>
<td></td>
<td>WinCC Explorer and select Properties</td>
</tr>
<tr>
<td></td>
<td>from the pop-up menu to open the</td>
</tr>
<tr>
<td></td>
<td>Computer List Properties dialog.</td>
</tr>
<tr>
<td></td>
<td>Click on the Properties button to open</td>
</tr>
<tr>
<td></td>
<td>the properties dialog of the local</td>
</tr>
<tr>
<td></td>
<td>computer.</td>
</tr>
<tr>
<td></td>
<td>In the Startup tab, the applications</td>
</tr>
<tr>
<td></td>
<td>to be activated with runtime are</td>
</tr>
<tr>
<td></td>
<td>selected. The Tag Logging Runtime</td>
</tr>
<tr>
<td></td>
<td>check-box must be selected.</td>
</tr>
<tr>
<td></td>
<td>The Computer Properties and Computer</td>
</tr>
<tr>
<td></td>
<td>List Properties dialogs can be closed</td>
</tr>
<tr>
<td></td>
<td>by clicking on OK.</td>
</tr>
</tbody>
</table>

![Computer properties dialog](image)
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 D 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 D on the Control.

- **Object Properties Dialog:** This is the default properties dialog of the Graphics Designer. It can be opened via a R 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creating a Process Value Archive</th>
</tr>
</thead>
</table>
| 1    | Create the tags to be archived in Tag Management.  
In this sample, the G64_ex_tlg_01, G64_ex_tlg_02 and G64_ex_tlg_03 tags are archived, which are supplied with values by the simulator.  
Create an additional tag of the Binary Tag type, which will store the current status of the archive. In this sample, this is the BINi_ex_tlg_00 tag. |
| 2    | Creation of a project function in the Global Script editor to start and stop the archiving.  
In this sample, this is the ZS_PA_Start function. Its functionality is described following this table. |
| 3    | Creation of a Process Value Archive in the Tag Logging editor.  
This is done via the Archive Wizard. In this sample, the archive has been named ZS_ProcessValueArchive_00. For the archiving, the G64_ex_tlg_01, G64_ex_tlg_02 and G64_ex_tlg_03 tags are selected. |
| 4    | Setting the properties of the Process Value Archive.  
The size of the archive is set to 1000 data records in the Archive Parameters tab.  
For the remaining options, the default settings are kept. |
## Step 5: Setting the properties of the Archive Tags

For each of the three tags, cyclic-selective is selected as the Archiving Type in the Archive Tag tab. This type of archiving gives you the option to set a Start Event and a Stop Event in the Events tab. In this sample, the previously created project function ZS_PA_Start is set as the Start Event. For the remaining options, the default settings are kept.

### Project Function ZS_PA_Start

```c
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

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

![Properties of WinCC Online Trend Control](image)
In the **Trends** tab, the trend profiles to be displayed are specified in detail. Three trends are created. The `G64_ex_tlg_01` to `G64_ex_tlg_03` tags of the `ZS_ProcessValueArchive_00` archive are assigned to these trends. The color of the three trends is set to black and the **Display Type** is set to **Show only Dots**. For the remaining options, the default settings are kept. The properties dialog of the Control can be closed by clicking on **OK**.

![Properties of WinCC Online Trend Control](image)
### Procedure: Configuring the Trend Display

4. Specific properties settings of the individual trends. For this purpose, an expanded properties dialog is available. This dialog is opened via a \(\text{Edit}\) on the Control. The properties dialog described previously, on the other hand, is opened via a \(\text{Edit}\) on the Control while the CTRL key is pressed.

The expanded properties dialog contains in addition to the already mentioned **General Information** and **Trends** tabs five additional tabs. In this sample, settings are made only in the **Value Axis** tab.

In the **Trend** field, set the entry `Trend_G64_ex_tlg_01` to define the properties of this trend. In the **Label** field, the text `Trend1` is entered. The **Range Selection** is not be performed automatically, but is set from -50 to 50. For the remaining options, the default settings are kept.

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 **OK**.
Configuration of the Toolbar and Status Bar

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuration of the Toolbar and Status Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create an internal tag of the Binary Tag type. In this sample, this is the BINi_ex_tlg_06 tag.</td>
</tr>
<tr>
<td>2</td>
<td>To control the update, a Smart Object (\rightarrow) Status Display is configured. In this sample, the Status Display5 object is used. Via its configuration dialog, the object is connected to the BINi_ex_tlg_06 tag and triggered upon change. The stati 0 and 1 are created. In this sample, the bitmaps stop tlg.bmp and stop go tlg.bmp are assigned to these stati. The object’s configuration dialog can be exited by clicking on OK.</td>
</tr>
<tr>
<td>3</td>
<td>For the just configured Status Display5 object, create a C-Action at Events (\rightarrow) Mouse (\rightarrow) Press Left. This C-Action simulates the pressing of the Stop/Go button of the Control’s standard toolbar. In addition, the status of the BINi_ex_tlg_06 tag is inverted to display the changed status of the Control’s update. A tag value of zero corresponds to an activated update. The status of the BINi_ex_tlg_06 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 direct connection at Events (\rightarrow) Miscellaneous (\rightarrow) Open Picture of the Picture Object ex_3_chapter_01a.pdl. This sets the status of the tag to 0.</td>
</tr>
</tbody>
</table>
### Step 4: Procedure: Configuring the Trend Display

4. As described in step 2, configure a second Smart Object ➔ Status Display. In this sample, this is the Status Display6 object. This object is used to control the archiving.

   This object is connected to the BINi_ex_tlg_00 tag created in the previous section. Correspondingly, different bitmaps are used (Archive.bmp / Archive inv.bmp).

   At Events ➔ Mouse ➔ Press Left, a C-Action is created. This action inverts the BINi_ex_tlg_00 tag. This tag is used to display the changed status of the archiving and to forward this information to the archive via the project function ZS_PA_Start.

5. 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.

   For the implementation, four Windows Objects ➔ Buttons are configured; in this sample, these are the Button4, Button7, Button8 and Button11 objects.

   For each of these objects, a C-Action is created at Events ➔ Mouse ➔ Mouse Action. These actions simulate the pressing of the buttons on the standard toolbar.

   In addition, a Smart Object ➔ Graphic Object is required that places itself over these buttons and makes them inoperational in case the update is started. In this sample, this is the Graphic Object2. This object displays the four buttons in an inoperational status (Pfeile dis.bmp). At Properties ➔ Miscellaneous ➔ Display, create a Dynamic Dialog. This dialog controls the visibility of the object dependent on the BINi_ex_tlg_06 tag, which contains information about the update of the Control.

6. To display the status bar, two Windows Objects ➔ Buttons are configured; in this sample, these are the Button5 and Button6 objects.

   For the text display, Buttons are used since they can easily be equipped with a 3D border. For this purpose, therefore, no additional objects are required.

   For Button5, create a C-Action at Properties ➔ Font ➔ Text. This action either returns the text Archiving Started or Archiving Stopped to the property dependent on the BINi_ex_tlg_00 tag. A C-Action instead of an equally applicable Dynamic Dialog is used to realize a language switch.

   Proceed in the same manner as just described for Button6 with the BINi_ex_tlg_06 tag.

| Update started... | Archiving stopped... |

**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 "apdefasp.h"
void OnButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszProper)
{
TlgTrendWindowPressStartStopButton("ZS_ProcessValueArchive_00");
SetTagBit("BINi_ex_tlg_06", (SHORT)1); 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 "apdefasp.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)

```c
#include <apedefap.h>
char* _rein(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    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 stopped... ");
        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.pdl)

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

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

In the Events dialog box, you enter those situations which trigger archiving.
Configuration of the Table Display

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuration of the Table Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a new picture in the Graphics Designer. In this sample, this is the <code>ex_3_chapter_01b.pdl</code> picture.</td>
</tr>
<tr>
<td>2</td>
<td>Configuration of the Control used for displaying the table. This is the WinCC Online Table Control. It is selected from the Object Palette’s Control selection menu and then placed in the picture.</td>
</tr>
</tbody>
</table>
### Step 3: Procedure: Configuration of the Table Display

After placing the Control in the picture, the *WinCC Online Table Control Properties* dialog is automatically opened.

In the *General Information* tab, you can specify the Control’s title and how it is labeled. In this sample, the *Display* check-box is deselected. A *Window Title* is still entered. In the *C-Actions* created later, this window title is used to reference the corresponding Control. The name of the previously created archive `AZ_ProcessValueArchive_00` is used.

Via the *Color* button, the *Background Color* of the table window is set to white.

In the *Display* field, all check-boxes are deselected in this sample. As a result, no *Toolbar* and no *Status Bar* is displayed.

![Properties of WinCC Online Table Control](image)

<table>
<thead>
<tr>
<th>General</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window Title:</td>
<td><em>Display</em></td>
</tr>
<tr>
<td><code>AZ_ProcessValueArchive_00</code></td>
<td></td>
</tr>
<tr>
<td>Open Screen:</td>
<td></td>
</tr>
<tr>
<td><em>Load archive data</em></td>
<td></td>
</tr>
<tr>
<td>Background Color</td>
<td></td>
</tr>
<tr>
<td><em>Color...</em></td>
<td></td>
</tr>
</tbody>
</table>

- **Display**
  - Status Bar
  - Toolbar
  - Common Time Column
  - Sizeable
4. **Procedure: Configuration of the Table Display**

In the *Columns* tab, the columns to be displayed are specified in detail. For this sample, only one column is needed.

One column has already been created. This column is renamed to `Row_G64i_ex_tlg_08`.

Via the *Selection* button, the *Archive Tag* to be displayed can be assigned to the column. In this sample, the *Archive Tag* `G64i_ex_tlg_08` of the previously created `AZ_ProcessValueArchive_00` archive is assigned to the column.

The properties dialog of the Control can be closed by clicking on *OK*.
### Step 5

**Procedure: Configuration of the Table Display**

Specific properties settings of the column. For this purpose, an expanded properties dialog is available. This dialog is opened via a `D` on the Control. The properties dialog described previously, on the other hand, is opened via a `F` D on the Control while the CTRL key is pressed.

The expanded properties dialog contains in addition to the already mentioned **General Information** and **Columns** tabs three additional tabs. In this sample, settings are made only in the **Column** tab.

The **Format** of the **Time Display** is changed to `hh:mm:ss`. The **Orientation** of the **Time Display** and the **Value Display** is set to **Centered**. In the **Time Selection** field, the **Time Range** check-box remains selected, but the set **Range** is changed to `1 X 1 Hour`.

For the remaining options, the presettings are kept. The properties dialog of the Control can be closed by clicking on **OK**.

![Properties of WinCC Online Table Control](image)

### Step 6

Writing to a tag if the set value is exceeded. This is done via a C-Action at the Control at `R Properties Geometry Position X`. The C-Action has been created at the property, since a trigger is required. The property itself is not made dynamic.
### Configuration of the Trend Display

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuration of the Trend Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Configuration of the Control used for displaying the trend profile. This is the WinCC Online Trend Control. It is selected from the Object Palette's Control selection menu and then placed in the picture. After placing the Control in the picture, its configuration dialog will be opened automatically. In the General Information tab, specify that the Control is to be displayed without a title bar. For this purpose, the Display check-box is deselected. In the Data Supply field, Online Tag 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. Via the Color button, the Background Color of the trend window is set to white. In the Display field, all check-boxes are deselected.</td>
</tr>
</tbody>
</table>

![Properties of WinCC Online Trend Control](image-url)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuration of the Trend Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>In the <em>Trends</em> tab, the trend profile to be displayed is specified in detail. The trend is renamed to <em>Trend_G64i_ex_tlg_04</em>. The <em>Color</em> of the trend is set to <em>blue</em>. As the <em>Display Type</em>, <em>Fill Area Interpolate Trend</em> is set. Via the <em>Selection</em> button, the <em>Tag Configuration</em> dialog is opened. In this dialog, the tag to be displayed is set. In this case, this <em>not</em> an <em>Archive Tag</em>, but a so-called <em>Online Tag</em> (an internal or external tag). In addition, a <em>Cycle Time</em> is set. In this sample, the <em>G64i_ex_tlg_04</em> tag with a cycle of <em>500 ms</em> is set. The properties dialog of the Control can be closed by clicking on <em>OK</em>.</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
3. Colored identification of a limit value violation. This can only be done in the expanded properties dialog. This dialog is opened via a D on the Control. The expanded properties dialog contains in addition to the already mentioned General Information and Trends tabs five additional tabs.

In the Value Axis tab, the Automatic check-box is deselected in the Range Selection field. The range is set fixed from -100 to 100.

In the Limit Values tab, a High Limit Value is configured. This value is set to 55. The color of the limit value is set to red.

The properties dialog of the Control can be closed by clicking on OK.
Configuration of the Toolbar and Status Bar

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

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

**Update started...**
C-Action at the WinCC Online Table Control (Control1)

```c
#include <apdsfap.h>
long _main(char* lpzPictureName, char* lpzObjectName, char* lpzPropertyName)
{
    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)

```c
#include <apdsfap.h>
void OnButtonDown(char* lpzPictureName, char* lpzObjectName, char* lpzPropertyName)
{
    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)

```c
#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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create two internal tags of the Binary Tag type. In this sample, these are the BINI_ex_tlg_06 and FLAG_TableGetOutputValue tags.</td>
</tr>
</tbody>
</table>
| 2    | Creation of a project function that instructs Tag Logging 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. In this sample, the functions EnumerateSuperArchiveData and GetArchiveDataCallback are used. The sample uses two external C variables.  
  - extern int dwSize  
  - extern WORD wOffset  
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 C-Action at Events → Miscellaneous → Open Picture of the start picture ex_0_startpicture_00.PDL. In the sample, this function is called CreateExternal. |
Step | Procedure: Implementation in the Graphics Designer
--- | ---
3 | Create a new picture, in the sample this is the `ex_3_chapter_01b.pdl` picture. The user-defined is to display 10 lines. For the display of the data in the first column, 10 *Standard Objects* `Static Texts` are used, which display the date and time. For the additional columns, *Smart Objects* `I/O Fields` are used.

As the object names in the first column, the names `Static Text1` to `Static Text10` are used, where the number defines the line. The numbering is performed from bottom to top, since the last line contains the latest data.

The `I/O Fields` use a number code as name. The first digit indicates the column number, the second number the line number.

<table>
<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>

4 | For each *Static Text*, a *C-Action* is created at *Properties* `Font` `Text`. 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 `FLAG_TableGetOutputValue` tag. The status of this tag changes, if the archive has received new data and this data has been read.

In the same manner, create a *C-Action* for each *I/O Field* at *Properties* `Output/Input` `Output Value`. This action has also the task to read the data record assigned to an object from the Callback function.

5 | To control the update, a *Smart Object* `Status Display` is configured. In this sample, the `Status Display3` object is used.

Via its *configuration dialog*, the object is connected to the `BINi_ex_tlg_06` tag and triggered upon change. The stati `0` and `1` are created and corresponding pictures assigned to each status. In this sample, the bitmaps *stop go tlg.gif* and *stop tlg.gif* are used.
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
</table>
| 6    | **At Events** → **Mouse** → **Press Left**, create a **C-Action** that inverts the status of the BINi_ex_tlg_06 tag. The status **TRUE** means that the update has been started.  
   The status of this tag is **TRUE** 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 **C-Action** at **Events** → **Miscellaneous** → **Open Picture** of the Picture Object ex_3_chapter_01c.pdl. This **C-Action** sets the status of the tag to **TRUE** and reads the archive one time. |
| 7    | **For the Statusdisplay3**, create a **C-Action** at **Properties** → **Geometry** → **Width**. This action, dependent on the status of the BINi_ex_tlg_06 tag, reads the archive and inverts the **FLAG_TableGetOutputValue** tag to trigger the update of the table. This **C-Action** is triggered upon the change of the G64i_ex_tlg_07 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 G64i_ex_tlg_07 tag is set to a value that cannot be reached by the process value to be archived after every run of the **C-Action**. |
| 8    | **To navigate in the archive while the update is stopped, navigation buttons are needed.**  
   For the implementation, four **Windows Objects** → **Buttons** are configured; in this sample, these are the **Button4**, **Button7**, **Button8** and **Button11** objects. |
| 9    | **At Events** → **Mouse** → **Mouse Action**, **C-Actions** are created. These actions write a new value to the external C variable **dwOffset**. In addition the trigger tag **FLAG_TableGetOutputValue** is inverted to achieve an update of the display.  
   In addition, a **Smart Object** → **Graphic Object** 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 **Graphic Object2**. The bitmap displayed by this object shows the four buttons in an inoperational status (**Pfeile dis.bmp**). **At Properties** → **Miscellaneous** → **Display**, create a **tag connection** to the BINi_ex_tlg_06 tag that is triggered upon change. |
#include "apdevap.h"

BOOL EnumerateSuperArchiveData()
{
    extern DWORD  dwSize;
    BOOL  fRet;
    TLG_GETARCHIVEDATA  GAD;
    CNTR_ERROR  Error;
    LPTSTR  lpszArchiveName = "ZK_ProcessValueArchive_02";
    LPTSTR  lpszVarName1 = "MaximumValue";
    LPTSTR  lpszVarName2 = "MinimumValue";
    LPTSTR  lpszVarName3 = "MeanValue";
    SYSTEMTIME  sysFrom;
    SYSTEMTIME  sysTo;
    time_t  Time;
    struct timebang  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,(FVOID)0);

    if (TLGConnect(NULL, &Error) == FALSE) {
        printf("Error: %s\n", Error.szErrorText);
        return FALSE;
    }

    else {
        fRet=TLGGetArchiveData(lpszArchiveName, lpszVarName1,
            sysFrom, sysTo, GetArchiveDataCallback, (FVOID)1, 0, &Error);
        if (fRet==FALSE)
            printf("Error: %s\n", Error.szErrorText);

        fRet=TLGGetArchiveData(lpszArchiveName, lpszVarName2,
            sysFrom, sysTo, GetArchiveDataCallback, (FVOID)2, 0, &Error);
        if (fRet==FALSE)
            printf("Error: %s\n", Error.szErrorText);

        fRet=TLGGetArchiveData(lpszArchiveName, lpszVarName3,
            sysFrom, sysTo, GetArchiveDataCallback, (FVOID)3, 0, &Error);
        if (fRet==FALSE)
            printf("Error: %s\n", Error.szErrorText);

        Call(&GAD,(FVOID)4);
        dwSize=GAD.deFlags;
        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**

```c
BOOL GetArchiveDataCallback (PTLG_GETARCHIVEDATA pGAD, PVOID lpUser) {
    static int i1 = 0;
    static int i2 = 0;
    static int i3 = 0;
    WORD vRecordNumber;
    WORD vColumnNumber;

    static TLG_GETARCHIVEDATA GAD1[100];
    static TLG_GETARCHIVEDATA GAD2[100];
    static TLG_GETARCHIVEDATA 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_GETARCHIVEDATA)*100);
        memset (&GAD2[0], 0, sizeof(TLG_GETARCHIVEDATA)*100);
        memset (&GAD3[0], 0, sizeof(TLG_GETARCHIVEDATA)*100);
        } //if (User==0)

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

    if (User==7)
    {
        vRecordNumber=lpGAD->stTime.vYear;
        vColumnNumber=lpGAD->stTime.vMonth;
        vColumnNumber=lpGAD->stTime.vDay;

        switch (vColumnNumber) {
        case 0 : lpGAD->stTime.vYear = GAD1[vRecordNumber].stTime.vYear;
        lpGAD->stTime.vMonth = GAD1[vRecordNumber].stTime.vMonth;
        lpGAD->stTime.vDay = GAD1[vRecordNumber].stTime.vDay;
        lpGAD->stTime.vHour = GAD1[vRecordNumber].stTime.vHour;
        lpGAD->stTime.vMinute = GAD1[vRecordNumber].stTime.vMinute;
        lpGAD->stTime.vSecond = GAD1[vRecordNumber].stTime.vSecond;
        break;
        case 1 : lpGAD->dvalue=GAD1[vRecordNumber].dvalue;
        break;
        case 2 : lpGAD->dvalue=GAD2[vRecordNumber].dvalue;
        break;
        case 3 : lpGAD->dvalue=GAD3[vRecordNumber].dvalue;
        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 IO 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

```c
#include "pldefap.h"
char* main(char* lppszPicturesName, char* lpszObjectName, char* lpzztPropert {

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=stoi(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 %02d %02d %02d %02d"
GAD.stTime.wDay, GAD.stTime.wMonth, GAD.stTime.wYear, GAD.stTime.wMonth, GAD.stTime.wYear, 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

```c
#include "apdfap.h"
void_data_main(chark* lpszPictureName, chark* lpszObjectName, chark* lpszProperty
{
  extern dwSize;
  int nRecordNumber;
  WORD wColumnNumber;
  extern WORD vOffset;
  char szObject[];
  TLG_GETARCHIVEDATA GAD;
  FVOID lpUser = 7;
  strcpy(szobject, ":");
  sprintf(szobject, ":%c", lpszObjectName[0]);
  wColumnNumber = (WORD)atoi(szobject);
  nRecordNumber = dwSize-atol(lpszObjectName+1)+vOffset;
  if (nRecordNumber(0) return 0;
  GAD stTime = (WORD)nRecordNumber;
  GAD stTime.wMonth = 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creating a Process Value Archive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create four tags. In this sample, these are the BIni_ex_tlg_m1, BIni_ex_tlg_m2 and BIni_ex_tlg_m3 tags of the Binary Tag type and the U08i_ex_tlg_00 tag of the Unsigned 8-Bit Value type.</td>
</tr>
<tr>
<td>2</td>
<td>Create a new Process Value Archive using the Archive Wizard. In this sample, the archive has been named ZK_ProcessValueArchive_03. For archiving, the BIni_ex_tlg_m1, BIni_ex_tlg_m2 and BIni_ex_tlg_m3 tags are selected.</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
</tr>
</tbody>
</table>
Step 4

Procedure: Creating a Process Value Archive

In the properties dialog of the first process tag, in the Archive upon field of the Parameters tab, select the entry Every Signal Change. In the Name of the Status field, Motor Off is entered for Status 0 and Motor On for Status 1. As the Cycle, Archiving 1*500 ms is set.

For the remaining options, the default settings are kept.
For the remaining archive tags, the same settings are performed.
Configuration of the Table Display

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuration of the Table Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a new picture, in the sample this is the <code>ex_3 chapter_01d.pdl</code> picture.</td>
</tr>
<tr>
<td>2</td>
<td>Configuration of the Control used for displaying the table. This is the WinCC Online Table Control. It is selected from the Object Palette’s Control selection menu and then placed in the picture.</td>
</tr>
<tr>
<td>3</td>
<td>After placing the Control in the picture, the WinCC Online Table Control Properties dialog is automatically opened. In the General Information tab, you can specify the Control’s title and how it is labeled. As the window title, Status is entered. Via the Color button, the Background Color of the table window is set to white. In the Display field, the Sizeable and Toolbar check-boxes are deselected.</td>
</tr>
</tbody>
</table>
Step 4: Procedure: Configuration of the Table Display

In the *Columns* tab, the columns to be displayed are specified in detail. For this sample, three columns are needed.

One column has already been created. It is renamed to Motor1.

Via the *Selection* button, the *Archive Tag* to be displayed can be assigned to the column. In this sample, the *Archive Tag* `BINi_ex_tlg_m1` of the previously created `ZK_ProcessValueArchive_03` archive is assigned to the column.

Add two more columns. They are assigned the `BINi_ex_tlg_m2` and `BINi_ex_tlg_m3` tags, and their column names and colors are adapted. The *Display Visible* check-box is deselected for all three columns.
Step | Procedure: Configuration of the Table Display
--- | ---
5 | Specific properties settings of the column. For this purpose, an expanded properties dialog is available. This dialog is opened via a D on the Control. The properties dialog described previously, on the other hand, is opened via a D on the Control while the CTRL key is pressed.

The expanded properties dialog contains in addition to the already mentioned General Information and Columns tabs three additional tabs. In the Column tab, the following settings are made.

The Format of the Time Display is set to hh:mm:ss. The Orientation of the Time Display and the Value Display is set to Centered and the decimal places to 0. In the Time Selection field, the Time Range check-box remains selected, but the set Range is changed to 1 X 1 Hour.

![Properties of WinCC Online Table Control](image)
## Procedure: Configuration of the Table Display

**Step 6**
In the *Toolbar* tab, the following check-boxes are selected at the *Key Functions* sub-entry:

- First Data Record
- Previous Data Record
- Next Data Record
- Last Data Record
- Select Time Range
- Start/Stop the Update

### Properties of WinCC Online Table Control

![Properties of WinCC Online Table Control](image)

**Step 7**
Since only one column is displayed, the font size can be set to 13.5 in the *Font* tab for increased legibility.

For the remaining options, the presettings are kept. The properties dialog of the Control can be closed by clicking on *OK*.

### Implementation in the Graphics Designer

**Step 1**
The motors to be displayed each consist of a *Standard Object ➔ Circle*, a *Standard Object ➔ Polygon* and a *Standard Object ➔ Static Text*. The background color of the circle is changed according to the motor status via a *Dynamic Dialog*.

These three objects are grouped. In the sample, this results in the creation of the *Group1*, *Group2* and *Group3* objects. Each of these objects receives a *direct connection* at *Events ➔ Mouse ➔ Mouse Action* that writes the number of the motor to the *U08i_ex_tlg_00* tag and a *C-Action at Events ➔ Mouse ➔ Press Left* that makes the current column visible (or the other columns invisible).
### Step 2: Procedure: Implementation in the Graphics Designer

2. Two Windows Objects \(\rightarrow\) Buttons are assigned to each motor. These buttons control via direct connections the tags assigned to the individual motors.

### Step 3: Implementation in the Graphics Designer

3. To identify the currently selected motor, a Standard Object \(\rightarrow\) Rectangle is assigned to each.

   - **At Properties** \(\rightarrow\) **Styles** \(\rightarrow\) **Line Style**, the dotted line is selected and at **Properties** \(\rightarrow\) **Styles** \(\rightarrow\) **Fill Pattern**, the Transparent pattern.
   - **At Properties** \(\rightarrow\) **Miscellaneous** \(\rightarrow\) **Display**, a Dynamic Dialog each is used to make the Rectangle visible only if the content of the U08i_ex_tlg_00 tag agrees with the own object number.

### C-Action at Motor1 (Group 1)

```c
#include "acdfsop.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszObjectNumber)
{

    /\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.pdl)

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a New Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Tag Logging editor from the WinCC Explorer. Create a new Timer by on the corresponding entry in the navigation window.</td>
</tr>
</tbody>
</table>
2. The properties dialog of the new Timer is displayed.

As the Name of the Timer, this sample uses 1min Start 0:0. As the Base, select 1 Minute with a Factor of 1. The Factor enables you to configure timers of, for example, four or six minutes. In the Starting Point of the Cycle field, select the Enter the Starting Point of the Cycle check-box. In each entry field of the cycle, 0 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.

Exit the dialog box by clicking on OK.

3. In the right window, the symbol of the newly created timer will be displayed in addition to the default timers.

<table>
<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>1 min Start 0:0</td>
<td>1 minute</td>
<td>1</td>
<td>02/03/98 09:08:36 AM</td>
</tr>
<tr>
<td>1 min Start 10:30</td>
<td>1 minute</td>
<td>1</td>
<td>02/03/98 09:10:01 AM</td>
</tr>
<tr>
<td>1 hour Start 0:0</td>
<td>1 hour</td>
<td>1</td>
<td>02/04/98 03:59:48 PM</td>
</tr>
</tbody>
</table>
## Creating a Process Value Archive

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

![Properties of process tag](image)

For the remaining archive tags, the same settings are made as have just been performed.
Implementation in the Graphics Designer

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

![Properties of WinCC Online Table Control](image)
Step 4

Procedure: Configuration of the Table Display

In the Columns tab, the columns to be displayed are specified in detail. For this sample, three columns are needed.

One column has already been created. This column is renamed to G64_ex_tlg_01.

Via the Selection button, the Archive Tag to be displayed can be assigned to the column. In this sample, the Archive Tag G64_ex_tlg_01 of the previously created ZK_ProcessValueArchive_01 archive is assigned to the column.

Add two more columns. They are assigned the BIni_ex_tlg_m2 and BIni_ex_tlg_m3 tags, and their column names and colors are adapted.
5 Specific properties settings of the column. For this purpose, an expanded properties dialog is available. This dialog is opened via a D on the Control. The properties dialog described previously, on the other hand, is opened via a D on the Control while the CTRL key is pressed. The expanded properties dialog contains in addition to the already mentioned General Information and Columns tabs three additional tabs. In the Column tab, the following settings are made.

The Format of the Time Display is set to hh:mm:ss. The Orientation of the Time Display and the Value Display is set to Centered. The Time Range is set to 1 x 1 Hour.

For the remaining options, the presets are kept. The properties dialog of the Control can be closed by clicking on OK.

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

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

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

![Properties of WinCC Online Trend Control](image)
### Step 4: Procedure: Configuration of the Trend Display

In the *Trends* tab, the trends to be displayed are specified in detail. For this sample, only one trend is needed. One trend has already been created. It is renamed to *Trend1*.

Via the *Selection* button, the *Archive Tag* to be displayed can be assigned to the trend. In this sample, the *Archive Tag G64_ex_tlg_04* of the previously created *ZK_ProcessValueArchive_04* archive is assigned to the trend. The *Display Type* of the trend is set to *Stepped Trend*.

![Properties of WinCC Online Trend Control](image)

**General** tab:
- **Trends**: Trend1
- **Name**: Trend1
- **Display**: Visible
- **Selection of Archives/Tags**: 
  - *ZK_ProcessValueArchive_04G64_ex*
- **Display type**: Stepped trend

**OK** | **Cancel** | **Apply** | **Help**
### Procedure: Configuration of the Trend Display

| Step | Specific properties settings of the trend. For this purpose, an expanded properties dialog is available. This dialog is opened via a \(\text{D}\) on the Control. The properties dialog described previously, on the other hand, is opened via a \(\text{D}\) on the Control while the CTRL key is pressed. The expanded properties dialog contains in addition to the already mentioned General Information and Trends tabs five additional tabs. In the Time Axis trend, the following settings are made. The Coarse Grid and Fine Grid check-boxes are deselected and the Time Format is set to \(hh:mm:ss\). In the Time Selection field, the Time Range check-box is deselected. |

### Properties of WinCC Online Trend Control

<table>
<thead>
<tr>
<th>Curves</th>
<th>General</th>
<th>Font</th>
<th>Toolbar</th>
<th>Time Axis</th>
<th>Value Axis</th>
<th>Limit values</th>
</tr>
</thead>
</table>

- **Trend:** Trend1
- **Label:**

- **Display**
  - [ ] Coarse Scaling
  - [ ] Fine Scaling

- **Time**
  - \(HHmm:ss\)

- **Selection of Time**
  - **Date:**
    - From: 6/22/99
    - To: 6/22/99
  - **Time:**
    - From: 9:59:35 AM
    - To: 10:00:35 AM

- **Time range:**
  - [ ]

- **Factor**
  - 1

- **Area**
  - [ ]

- **Number of measurement pts:**
  - 120

---
**Procedure: Configuration of the Trend Display**

6. In the Value Axis tab, all check-boxes are deselected from the Trend field. In the Area Selection field, the Automatic check-box is deselected and the values from -100 to 100 are entered.

For the remaining options, the presets are kept. The properties dialog of the Control can be closed by clicking on **OK**.
Project Function for Exporting the Archive

```c
void ActionForExportingArchive(LPCTSTR lpszArchivNameReturn, LPCTSTR lpszVar
{
    int iRet = 0;
    CNV_ERROR Err;
    char szProj[MAX_PATH];
    char szFileName[MAX_PATH];
    LPCTSTR lpszArchivName = "FDE#HDZK_ProcessValueArchive_04#G641.exe_tlg_04";
    char szFileName[MAX_PATH] = "";
    LPCTSTR lpszFileName;
    TLG_IO_BACKUP_SELECT ibs;
    DWORD dwSize;
    time_t Time;
    struct tm* TimeStruct;
    int nPathLen, nFileLen;

    DMGetRuntimeProject( szProj, MAX_PATH, &Err);
    nPathLen=strlen(szProj);
    nFileLen=strlen(strchr(szProj,'\')+1);

    sprintf(szFileName, "%s\", szFile, "ArchiveBackup.csv");
    lpszFileName=aszFileName[0];

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

    ibs.sysFrom.vYear = 1997;
    ibs.sysFrom.vMonth = 1;
    ibs.sysFrom.vDay = 1;
    ibs.sysFrom.vHour = 0;
    ibs.sysFrom.vMinute = 0;
    ibs.sysFrom.vSecond = 0;

    ibs.sysTo.vYear = (WORD)(TimeStruct->tm_year+1900);
    ibs.sysTo.vMonth = (WORD)(TimeStruct->tm_mon+1);
    ibs.sysTo.vDay = (WORD)(TimeStruct->tm_mday);
    ibs.sysTo.vHour = (WORD)(TimeStruct->tm_hour);
    ibs.sysTo.vMinute = (WORD)(TimeStruct->tm_min);
    ibs.sysTo.vSecond = (WORD)(TimeStruct->tm_sec);

    fRet = TLGConnect(NULL, &Err);
    if (fRet==FALSE) print("Error in TLGConnect[\n"");
        fRet = TLGGetBackupSize(lpszArchivName, dwSize, &ibs, TLG_BACKUP_EXPORT);
    if (fRet==FALSE) print("Error in TLGGetBackupSize[\n", Err.szErrorText);
        SetTagWord(0x6165_tlg_00, (WORD)dwSize);

    fRet = TLGBackup(lpszArchivName, lpszFileName, &ibs, TLG_BACKUP_EXPORT, TLG);
    if (fRet==FALSE) print("Error in TLGBackup[\n", Err.szErrorText);
        SetTagBit(0x6165_tlg_00, 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 BNI_ex_tlg_09, which makes the export visible.
## Implementation in the Graphics Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, create three tags of the <strong>Binary Tag</strong> type. In this sample, the <code>BINi_ex_tlg_06</code>, <code>BINi_ex_tlg_08</code> and <code>BINi_ex_tlg_09</code> tags are used. Additionally, the <code>U16i_ex_tlg_00</code> tag of the <strong>Unsigned 16-Bit Value</strong> type is required.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 3    | To control the archiving, a **Windows Object** ➔ **Button** is configured. In this sample, this is the `Button16` object.  

At **Events ➔ Mouse ➔ Press Left**, a **C-Action** is created that inverts the status of the `BINi_ex_tlg_08` tag and calls the project function `LockUnlockArchive`. The binary tag is used to store the current status of the archive. |
| 4    | Configure a new picture that will be displayed when the archive if exported. In the sample, this is the `ex_5_window_03.PDL` picture.  

This picture contains a **Standard Object** ➔ **Static Text** which displays a text via a **C-Action**. This text consists of a fixed part and the number value of the `U16i_ex_tlg_00` tag. This tag contains the size of the exported data. Additionally, the picture contains a **Windows Object** ➔ **Button** and a **Smart Object ➔ Graphic Object** that both, via a **direct connection** at **Events ➔ Mouse ➔ Mouse Action**, switch the constant 0 to the `BINi_ex_tlg_09` tag. |
| 5    | In the initial picture, configure a **Smart Object ➔ Picture Window**; in the sample this is the `Picture Window1` object. At **Properties ➔ Miscellaneous ➔ Picture Name**, set the `ex_5_window_03.PDL` picture. At **Properties ➔ Miscellaneous ➔ Display**, create a tag connection to the `BINi_ex_tlg_09` tag. |
| 6    | To display the status bar, two **Windows Objects ➔ Buttons** are configured; in this sample, these are the `Button14` and `Button17` objects.  

For `Button17`, create a **Dynamic Dialog** at **Properties ➔ Font ➔ Text** that, dependent on the `BINi_ex_tlg_06` tag, either returns the text **Update Started** or **Update Stopped** to the property.  

For `Button14`, create a **Dynamic Dialog** at **Properties ➔ Font ➔ Text** that, dependent on the `BINi_ex_tlg_08` tag, either returns the text **Archive Enabled** or **Archive Locked** to the property. |
Project Function for Locking and Enabling the Archive

```c
void LockUnlockArchive (BOOL bLock)
{
    BOOL fRet;    /* Error */
    LPTSTR lpszArchiveName = "ZK_ProcessValueArchive_04";

    fRet = TLGConnect( NULL, &Error );
    if (fRet==FALSE)
        printf("Error in TLGConnect( )\n");
    else
        {
            fRet = TLGLockArchive (NULL, lpszArchiveName, bLock, &Error );
            if (fRet==FALSE)
                printf("Error in TLGLockArchive( )\n");
            Error.szErrorMessage;
        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 button. 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 status 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 status are realized using *C-Actions*.

**Creation of the Required Tags**

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of the Required Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a total of twelve tags of the <em>Unsigned 16-Bit Value</em> type in Tag Management. Four of these tags are used as event tags. In the sample, these are the <em>U16i_ex_alg_00</em>, <em>U16i_ex_alg_03</em>, <em>U16i_ex_alg_06</em> and <em>U16i_ex_alg_09</em> tags. Four other tags are used as status tags; in the sample, these are the <em>U16i_ex_alg_02</em>, <em>U16i_ex_alg_05</em>, <em>U16i_ex_alg_08</em> and <em>U16i_ex_alg_11</em> tags. The remaining tags, <em>U16i_ex_alg_12</em>, <em>U16i_ex Alg_13</em>, <em>U16i_ex Alg_14</em> and <em>U16i_ex_alg_15</em> in the sample, serve as acknowledgment tags.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Message Block Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the WinCC Explorer, open the Alarm Logging editor by clicking on it and then selecting Open from the pop-up menu.</td>
</tr>
<tr>
<td>2</td>
<td>Selection of the desired message blocks. This is done via a right-click on the Message Blocks entry and then selecting Message Blocks from the pop-up menu. The Configure Message Blocks dialog is opened.</td>
</tr>
<tr>
<td>3</td>
<td>Via the Add button, the dialog for adding blocks is opened for the selected system blocks entry, user text blocks entry or process value blocks entry.</td>
</tr>
</tbody>
</table>

If a block is selected with a checkmark in the Configure Message Blocks dialog, the buttons Remove and Properties become operational. The first button allows you to remove selected blocks, the second button allows you to configure the properties of the individual message blocks.
### Creation of Single Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of Single Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Alarm Logging 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 ( \mathbb{R} ), new lines can be added. 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 Single Message dialog. This dialog is opened via a ( \mathbb{R} ) on the appropriate message line.</td>
</tr>
</tbody>
</table>
### Step 2: Procedure: Message Block Configuration

Open the *Single Message* dialog for the first line as described in the previous step.

In the *Parameters* tab, select the message class *Error* and the message type *Failure*.

In the *This Message* field, select the check-boxes *will be archived* and *will be reported*.

In the *Connections* field, select as the *Event Tag* the tag `U16i_ex_alg_00`. As the *Event Bit*, 0 is entered. This means that the message will be generated if the first bit of the tag set takes on the status 1.

As the *Acknowledge Tag*, the tag `U16i_ex_alg_12` is selected and as the *Acknowledge Bit*, 0 is entered. In other words, if the message is acknowledged in runtime, the first bit of the tag set is set to 1.

As the *Status Tag*, the tag `U16i_ex_alg_02` is selected and as the *Status Bit*, 0 is entered. This setting means that the first bit of the tag set represents the *Came in/Went out Status* 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 *Acknowledge Status* of the message. If it is not acknowledged, the bit has the status 1, if it is acknowledged, the status 0.

A status tag of 16 bits can represent the status of 8 single messages. The Low Byte contains the *Came in/Went out Status* and the High-Byte the *Acknowledge Status*. 

![Image of Single Message dialog](image)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Text</th>
<th>Tag/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class: Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type: Failure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group: None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- is single acknowledgment only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- trigger horn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- will be archived</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- will be reported</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- is created on a negative edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- triggers an action</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Image of Connections field](image)
### Step 3

**Procedure: Message Block Configuration**

In the **Text** tab, the *Message Text Locking Error* and the *Point of Error Motor 1* are entered. No **Infotext** is used. The **Tag/Action** tab must not be filled out for this sample.

The settings made are applied by clicking on **OK**.

![Single message configuration](image)

---

4

The message just created monitors the first of the four motors. For the first motor, two more message lines are created.

The settings are made as described in step 2 to 3, but the *Event Bits*, *Acknowledge Bits* and *Status Bits* are adapted. Also, the *Message Texts Feedback Error* and *Bimetal Error* are used.

---

5

For the other three motors, three message lines each are also created.

Here, the *Event Tags*, *Acknowledge Tags* and *Status Tags* as well as the texts for the *Point of Error* must be adapted to each motor.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Type</th>
<th>Message Tag</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Lock Error</td>
</tr>
<tr>
<td>2</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
<tr>
<td>3</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Bimetal Error</td>
</tr>
<tr>
<td>4</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
<tr>
<td>5</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Bimetal Error</td>
</tr>
<tr>
<td>6</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
<tr>
<td>7</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Bimetal Error</td>
</tr>
<tr>
<td>8</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
<tr>
<td>9</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Bimetal Error</td>
</tr>
<tr>
<td>10</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
<tr>
<td>11</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
<tr>
<td>12</td>
<td>Error</td>
<td>Error</td>
<td>U16_ex_e03</td>
<td>Feedback Error</td>
</tr>
</tbody>
</table>
Color Scheme Configuration of the Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Color Scheme Configuration of the Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The configured single messages are of the message class <em>Error</em> and the message type <em>Failure</em>. Via a [ ] on the <em>Message Classes</em> entry, all available message classes will be displayed in the right window. Via a [ ] on the icon of the message class <em>Error</em>, all message types pertaining to this class will be displayed. Via a [ ] on the icon of the message type <em>Failure</em> or via a [ ] on it and then selecting properties from the pop-up menu, the <em>Type</em> dialog is opened.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Type dialog" /></td>
</tr>
<tr>
<td>2</td>
<td>In the <em>Type</em> dialog, a color scheme can be created for each message status. In this sample, the following color scheme is used: Came in: Text = Yellow, Background = Orange Went out: Text = Orange, Background = Light Gray Acknowledged: Text = White, Background = Orange</td>
</tr>
<tr>
<td>3</td>
<td>The configurations made in <em>Alarm Logging</em> are saved via the File [ ] <em>Save</em> menus.</td>
</tr>
</tbody>
</table>
### Implementation in the Graphics Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a new picture; in this project, this is the <code>ex_3_chapter_02</code> picture.</td>
</tr>
<tr>
<td>2</td>
<td>The individual motors are displayed via a <em>Standard Object</em> ➔ <em>Circle</em>, a <em>Standard Object</em> ➔ <em>Static Text</em> and a <em>Standard Object</em> ➔ <em>Polygon</em>. 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 status came in, went out and acknowledged. For this, a <em>C-Action</em> is created for the <em>Static Text</em> at <code>Properties ➔ Colors ➔ Font Color</code>, which changes the font color dependent on the current status of the status tag belonging to the motor. Likewise, a <em>C-Action</em> is created for the <em>Circle</em> at <code>Properties ➔ Colors ➔ Background Color</code> that performs the same task.</td>
</tr>
<tr>
<td>3</td>
<td>The occurrence of an error at a motor is simulated via a <em>Windows Object</em> ➔ <em>Check-Box</em>. At <code>Properties ➔ Geometry ➔ Number of Boxes</code>, 3 is entered. At <code>Properties ➔ Output/Input ➔ Selected Boxes</code>, a <em>Tag Connection</em> to the motor's corresponding event tag is created.</td>
</tr>
</tbody>
</table>

![Check-Box Diagram](image)
**Step** | **Procedure: Implementation in the Graphics Designer**
---|---
4 | To display the messages configured in *Alarm Logging*, a *WinCC Alarm Control* is used. It is selected from the Object Palette’s *Control* selection menu and then placed in the picture.

![Object Palette](image1.png)

5 | After placing the Control in the picture, its *configuration dialog* will be displayed automatically. This dialog can be closed by clicking on *OK*.

Open the Control’s *Properties* dialog. This dialog is displayed via a "C" on the Control. In the *General Information* tab, the *Selection* button is used to select the single messages created in *Alarm Logging* to be displayed by the Control.

![WinCC Alarm Control Properties](image2.png)
Step | Procedure: Implementation in the Graphics Designer
--- | ---
6 | Via a $\square$ on the system block *Number*, 2 check-boxes will be displayed in the right window. The *Start Value* is changed via a $\square$ on the default value 0 to 1 and the *Stop Value* to 12. This means that the Control only displays the single messages from number 1 to 12.

**Specify Selection**

- System blocks
  - Date
  - Time
  - Number
- Message class
- Text blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start value</td>
<td>1</td>
</tr>
<tr>
<td>Stop value</td>
<td>12</td>
</tr>
</tbody>
</table>

MSGNR $\geq$ 1 AND MSGNR $\leq$ 12

[OK] [Cancel]
### Step 7

**Procedure: Implementation in the Graphics Designer**

In the *Toolbar* tab, the buttons to be displayed in runtime are selected. In this sample, the following buttons are required: *Single Acknowledgment, Group Acknowledgment, Auto Scroll On/Off, Beginning of the List, End of the List, Next Message and Previous Message.*

![WinCC Alarm Control Properties](image)

### Step 8

In the *Message Blocks* tab, the columns are selected that will later be displayed in the message line. In this sample, system blocks are selected in the *Type* field using the ![ ](image). In the right window, *Date, Time and Number* are selected. For the *User Text Blocks* entry, *Message Text and Point of Error* are selected.

![WinCC Alarm Control Properties](image)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>In the Message Line tab, the previously selected Message Blocks are assigned to the message line. The Available Message Blocks fields lists the available columns. By pressing on the -&gt; button, each message block can individually be added to the message line. Via the &gt;&gt; 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 OK.</td>
</tr>
</tbody>
</table>

![WinCC Alarm Control Properties](image)

 existing message blocks:  elements of the message

- Number
- Date
- Time
- Message text
- Point of error

Please select the needed message blocks.
### Step 10

**Procedure: Implementation in the Graphics Designer**

Activation of *Alarm Logging Runtime*.

For this, **Ctrl R** on the *Computer* entry in the *WinCC Explorer* and select *Properties* from the pop-up menu to open the *Computer List Properties* dialog. Click on the *Properties* button to open the properties dialog of the local computer.

In the *Startup* tab, the applications to be activated with runtime are selected. The *Alarm Logging Runtime* check-box must be selected.

The *Computer Properties* and *Computer List Properties* dialogs can be closed by clicking on *OK*.

![Computer properties dialog](image)

*Note: The image shows the *Computer properties* dialog with the *Alarm Logging Runtime* check-box selected.*
C-Action at the Circle (Circle1)

```
#include "opcdtop.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropStr,
DWORD state;
stateGetTagDWORD("U16i_ex_alg_02");
if (((state&1)|| (state&2))|| (state&4))
    return 0x80FF;
else
    return 0xFFFFFFFF;
```

- This C-Action makes the Background Color property of the Circle assigned to the first motor dynamic.

- The status tag $U16i_{\text{ex}}_{\text{alg}}_{02}$ assigned to the first motor is read. The Low Byte of this tag contains the message status 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 ffffff).

- This C-Action is triggered upon the change of the status tag $U16i_{\text{ex}}_{\text{alg}}_{02}$.

C-Action at the Static Text (StaticText1)

```
#include "opcdtop.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropStr,
DWORD state;
stateGetTagDWORD("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 0xFFFFFFF;
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 $U16i_{\text{ex}}_{\text{alg}}_{02}$ assigned to the first motor is read. The Low Byte of this tag contains the message status 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 ffffff); 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 $U16i_{\text{ex}}_{\text{alg}}_{02}$.

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of the Required Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a total of six tags of the Unsigned 16-Bit Value type in Tag Management. Three of these tags contain the temperature values of the individual containers. In the sample, these are the U16i_ex_alg_t1, U16i_ex_alg_t2 and U16i_ex_alg_t3 tags. The remaining three tags each contain the pressure values. In the sample, these are the U16i_ex_alg_p1, U16i_ex_alg_p2 and U16i_ex_alg_p3 tags. Three additional tags of the Unsigned 16-Bit Value type are required that are used as status tags. In the sample, these are the U16i_ex_alg_01, U16i_ex_alg_04 and U16i_ex_alg_07 tags. One tag of the Unsigned 16-Bit Value type for the control of the central indicator is required; in the sample, this is the U16i_ex_alg_10 tag. Additionally, two tags of the Binary Tag type are required. In this sample, these are the BINi_ex_alg_00 and BINi_ex_alg_03 tags.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a new Message Window Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Alarm Logging editor.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Open the Configure Message Blocks dialog via a on the Message Blocks entry. In the dialog, select the Process Value Blocks list entry and open the Add Process Value Blocks... dialog by clicking on the Add button. In this dialog, a new process value block is added. The dialog is closed by clicking on OK.</td>
</tr>
<tr>
<td></td>
<td>Via a on the Process Value Blocks 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, Value was entered as the Name of the block and 5 characters as its Length.</td>
</tr>
<tr>
<td></td>
<td>By clicking on OK, the settings made in the Message Blocks and Configure Message Blocks dialogs are applied.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a new Message Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Via a `R on the Message Class entry, open the Add Message Class... dialog.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Add Message Class dialog" /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Properties" /></td>
</tr>
<tr>
<td>2</td>
<td>Via the -&gt; button, a new message class is added. Close the dialog box by clicking on OK.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Select the required message classes" /></td>
</tr>
<tr>
<td>3</td>
<td>Via a <code>R on the Message Class entry, all created message classes, even the newly added one, are displayed. Via a </code>R on its icon, the Configure Message Class dialog can be opened.</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Configure Message Class dialog" /></td>
</tr>
</tbody>
</table>
### Procedure: Creation of a new Message Class

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| 4    | In the **General Information** tab, as the **Name of the Class**, **Container Error** is entered. Via the add button, the **Add Message Type...** dialog is accessed. In this dialog, two message types from the left window are moved to the right window via the `- >` button. Close the dialog box by clicking on **OK**. If one of the new message types is selected in the **Message Types** field, its properties dialog can be opened via the properties button. As the name of the first message type, **Container Alarm** is entered. The color scheme of the individual message stati looks as follows:  
  - Came in: Text = black, Background = red  
  - Went out: Text = black, Background = green  
  - Acknowledged: Text = black, Background = orange  
As the name of the second message type, **Container Warning** is entered. The color scheme of the individual message stati looks as follows:  
  - Came in: Text = yellow, Background = blue  
  - Went out: Text = blue, Background = RGB(207,163,146)  
  - Acknowledged: Text = white, Background = blue |

![Configure message classes.](image)
### Step 5
**Procedure: Creation of a new Message Class**

In the *Acknowledgment* tab, the *Acknowledgment Came In* check-box is selected from the *Acknowledgment Philosophy* field. In the *Acknowledgment of Central Indicator* field, the *Separate Acknowledgment Key* radio-button is selected. As the *Tag*, the `U16i_ex_alg_10` 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 *Horn Acknowledgment* button.

#### Configure message classes.

<table>
<thead>
<tr>
<th>Acknowledgement Theory</th>
<th>Central Signaling Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgement Came in</td>
<td>Separate Ack. Key</td>
</tr>
<tr>
<td></td>
<td>By Means Of Single Ack.</td>
</tr>
<tr>
<td>Flashing On</td>
<td>Tag:</td>
</tr>
<tr>
<td></td>
<td><code>U16i_ex_alg_10</code></td>
</tr>
<tr>
<td>Messages Without Status &quot;Went Out&quot;</td>
<td>...</td>
</tr>
<tr>
<td>Single message with incoming</td>
<td></td>
</tr>
</tbody>
</table>

Specify the acknowledgement theory and choose a setting for how central signalling devices (horns, lights, etc.) should be acknowledged.

### Step 6
In the *Status Texts* tab, no additional settings are made. Close the dialog box by clicking on *OK*.
4.2.3 Limit Value Monitoring (Continuation)

Creation of Single Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of Single Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Via a R in the table window, 12 new lines are added.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Type</th>
<th>MessageTag</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>Failure</td>
<td>U16i_ex_alg_00</td>
</tr>
<tr>
<td>4</td>
<td>Error</td>
<td>Failure</td>
<td>U16i_ex_alg_03</td>
</tr>
<tr>
<td>5</td>
<td>Error</td>
<td>Failure</td>
<td>U16i_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>I1Hr_ex_alarm_06</td>
</tr>
</tbody>
</table>

The first of the newly added lines is selected via the R. Via a R on this line, the Single Messages dialog can be opened.

2 In the Parameters tab, select the message class Container Error and the message type Container Alarm. In the This Message field, the is single acknowledgment, controls the horn, will be archived and will be reported check-boxes are selected. In the Connections field, select as the Status Tag the tag U16i_ex_alg_01. As the Status Bit, 0 is entered. No Event Tag is set, since the message is generated by the limit value monitoring. Likewise, no Acknowledge Tag is set.
### Step 3

**Procedure: Creation of Single Messages**

In the **Text** tab, the *Message Text Critical Overpressure* and the *Point of Error Container 1* are entered. As the **Infotext**, *The pressure in container 1 has exceeded the critical value* is entered.

<table>
<thead>
<tr>
<th>Message Text</th>
<th>Critical Overpressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point of Error</td>
<td>Container 1</td>
</tr>
</tbody>
</table>

In the **Tag/Action** tab, a tag could be set for the **Value** 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.

The settings made are applied by clicking on **OK**.

<table>
<thead>
<tr>
<th>Process Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process value: 2</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 3</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 4</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 5</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 6</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 7</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 8</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 9</td>
<td>R</td>
</tr>
<tr>
<td>Process value: 10</td>
<td>R</td>
</tr>
</tbody>
</table>
The just created message monitors the pressure in the first of the three containers. For the first container, three more message lines are created. The settings are made as described in step 2, however, for the additional message of the message type **Container Error**, the **Message Text Critical Temperature** and a correspondingly changed **Infotext** are entered. In addition, two messages of the message type **Container Warning** are created, which have the **Message Texts Pressure Warning** and **Temperature Warning**. For these warnings, all checkboxes in the **This Message** field in the **Single Message** dialog of the **Parameters** tab are deselected. For all messages pertaining to container 1, the same status tag, but with an adapted status bit, is used.

For the other two containers, four messages each are also created. Here, the **Status Tags** and the texts for the **Point of Error** must be adapted to the respective containers.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Type</th>
<th>Message tag</th>
<th>Status tag</th>
<th>Status bit</th>
<th>Message text</th>
<th>Point of err</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_02</td>
<td>U1G_ez_ag_02</td>
<td>1</td>
<td>Feedback Error</td>
<td>Motor 2</td>
</tr>
<tr>
<td>7</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_03</td>
<td>U1G_ez_ag_03</td>
<td>2</td>
<td>Feedback Error</td>
<td>Motor 2</td>
</tr>
<tr>
<td>8</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_04</td>
<td>U1G_ez_ag_04</td>
<td>0</td>
<td>Lock Error</td>
<td>Motor 3</td>
</tr>
<tr>
<td>9</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_05</td>
<td>U1G_ez_ag_05</td>
<td>1</td>
<td>Feedback Error</td>
<td>Motor 3</td>
</tr>
<tr>
<td>10</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_06</td>
<td>U1G_ez_ag_06</td>
<td>2</td>
<td>Feedback Error</td>
<td>Motor 3</td>
</tr>
<tr>
<td>11</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_07</td>
<td>U1G_ez_ag_07</td>
<td>0</td>
<td>Lock Error</td>
<td>Motor 4</td>
</tr>
<tr>
<td>12</td>
<td>Ex</td>
<td>Fail</td>
<td>U1G_ez_ag_08</td>
<td>U1G_ez_ag_08</td>
<td>1</td>
<td>Feedback Error</td>
<td>Motor 4</td>
</tr>
<tr>
<td>13</td>
<td>Fail</td>
<td>Alarm Container</td>
<td>U1G_ez_ag_09</td>
<td>U1G_ez_ag_09</td>
<td>0</td>
<td>Critical Temperature</td>
<td>Container 1</td>
</tr>
<tr>
<td>14</td>
<td>Fail</td>
<td>Alarm Container</td>
<td>U1G_ez_ag_10</td>
<td>U1G_ez_ag_10</td>
<td>1</td>
<td>Critical Temperature</td>
<td>Container 1</td>
</tr>
<tr>
<td>15</td>
<td>Fail</td>
<td>Alarm Container</td>
<td>U1G_ez_ag_11</td>
<td>U1G_ez_ag_11</td>
<td>0</td>
<td>Critical Temperature</td>
<td>Container 2</td>
</tr>
<tr>
<td>16</td>
<td>Fail</td>
<td>Alarm Container</td>
<td>U1G_ez_ag_12</td>
<td>U1G_ez_ag_12</td>
<td>1</td>
<td>Critical Temperature</td>
<td>Container 2</td>
</tr>
<tr>
<td>17</td>
<td>Fail</td>
<td>Alarm Container</td>
<td>U1G_ez_ag_13</td>
<td>U1G_ez_ag_13</td>
<td>0</td>
<td>Critical Temperature</td>
<td>Container 3</td>
</tr>
<tr>
<td>18</td>
<td>Fail</td>
<td>Alarm Container</td>
<td>U1G_ez_ag_14</td>
<td>U1G_ez_ag_14</td>
<td>1</td>
<td>Critical Temperature</td>
<td>Container 3</td>
</tr>
<tr>
<td>19</td>
<td>Fail</td>
<td>Warning Container</td>
<td>U1G_ez_ag_15</td>
<td>U1G_ez_ag_15</td>
<td>0</td>
<td>Warning pressure</td>
<td>Container 1</td>
</tr>
<tr>
<td>20</td>
<td>Fail</td>
<td>Warning Container</td>
<td>U1G_ez_ag_16</td>
<td>U1G_ez_ag_16</td>
<td>1</td>
<td>Warning pressure</td>
<td>Container 1</td>
</tr>
<tr>
<td>21</td>
<td>Fail</td>
<td>Warning Container</td>
<td>U1G_ez_ag_17</td>
<td>U1G_ez_ag_17</td>
<td>0</td>
<td>Warning pressure</td>
<td>Container 2</td>
</tr>
<tr>
<td>22</td>
<td>Fail</td>
<td>Warning Container</td>
<td>U1G_ez_ag_18</td>
<td>U1G_ez_ag_18</td>
<td>1</td>
<td>Warning temperature</td>
<td>Container 2</td>
</tr>
<tr>
<td>23</td>
<td>Fail</td>
<td>Warning Container</td>
<td>U1G_ez_ag_19</td>
<td>U1G_ez_ag_19</td>
<td>0</td>
<td>Warning temperature</td>
<td>Container 2</td>
</tr>
<tr>
<td>24</td>
<td>Fail</td>
<td>Warning Container</td>
<td>U1G_ez_ag_20</td>
<td>U1G_ez_ag_20</td>
<td>1</td>
<td>Warning temperature</td>
<td>Container 3</td>
</tr>
</tbody>
</table>
## Configuration of the Limit Value Monitoring

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Configuration of the Limit Value Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If the Limit Value Monitoring (Analog Alarm) entry is not present in the navigation window, it must be loaded. This is done via the <strong>Options</strong> → <strong>Add Ins</strong> menus in Alarm Logging. In the displayed dialog, the entry for the limit value monitoring must be check-marked.</td>
</tr>
</tbody>
</table>

### Add Ins

Select the Add Ins that your project requires from here:

- **Add Ins:**
  - Analog Alarm

- **Description**

- This dialog box can also be displayed via the menu command "Extras" and the entry "Add Ins ..."

### OK Cancel

<table>
<thead>
<tr>
<th>2</th>
<th>Via a <strong>R</strong> on the Limit Value Monitoring entry and then selecting New from the pop-up menu, the <strong>Properties</strong> dialog of the tag is accessed. In this dialog, a new tag for the limit value monitoring can be set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>Procedure: Configuration of the Limit Value Monitoring</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>In this dialog, the U16i_ex_alg_t1 tag containing the temperature value of the first container is set as the <strong>Tag to be monitored</strong>. The check-box <strong>a message for all limit values</strong> is not selected. As the <strong>Delay Time</strong>, 0 is kept. Exit the dialog box by clicking on <strong>OK</strong>.</td>
</tr>
</tbody>
</table>

- **Properties**
  - **Tag to be monitored**: U16i_ex_alg_t1
  - **Number of limit values**: 
  - **a message for all limit values; message number**: 0

  *If selected, only one message can be used for all limit values of the tag. In runtime, one instance of this message will be created.*

  - **Delay**
    - **Delay time**: 0
    - **Delay Time**: Miliseconds, Minutes, Seconds, Hours

  *The shortest delay time is 250 milliseconds, the longest 24 hours. A delay time of 0 equals no delay time.*

  - **Please note that the settings made will be valid for all limit values of this tag.**

| 4    | In the right window, the icon of the tag to be monitored is displayed. Via a right-click on this icon and then selecting **New** from the pop-up menu, the **Properties** dialog of the limit value is opened. In this dialog, a new limit value can be assigned to the tag. |

- **Properties**
  - **New**
  - **Delete**
  - **Properties**
### Step 5: Procedure: Configuration of the Limit Value Monitoring

In the Limit Value field, the Upper Limit is set. In the Limit Value or Tag field, a limit value of 800 is entered. As the Hysteresis, 0 is kept. As the Message, the Number 14 is entered. This is the alarm message if the temperature in the first container is exceeded.

Exit the dialog box by clicking on OK.

For the same tag, a second limit value is specified. In the Limit Value field, an Upper Limit is set again. However, in the Limit Value or Tag field, a limit value of 500 is entered. As the Message, the Number 20 is entered. This is the warning message if the temperature in the first container is exceeded.

#### Properties

<table>
<thead>
<tr>
<th>Limit value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper limit</td>
<td></td>
</tr>
<tr>
<td>Lower limit</td>
<td></td>
</tr>
</tbody>
</table>

#### Deadband

<table>
<thead>
<tr>
<th>absolute</th>
<th>Hysteresis:</th>
</tr>
</thead>
<tbody>
<tr>
<td>in percent</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Message

| Number: | 14 |

### Step 6

The remaining five tags to be monitored are created as described in steps 2 and 3, each with two configured limit values.

Via a "Analog Alarm" on the Limit Value Monitoring entry, all created tags will be displayed.

- U18I_ex_aig_t1
- U18I_ex_aig_t2
- U18I_ex_aig_t3
- U18I_ex_aig_p1
- U18I_ex_aig_p2
- U18I_ex_aig_p3
- U18I_ex_aig_01
Implementation in the Graphics Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The simulation of the process values to be monitored is implemented via a Windows Object ➔ Slider Object each. In the sample, these are the Slider Object1 to Slider Object6. For Slider Object1, create a Direct Connection at Events ➔ Property Topics ➔ Miscellaneous ➔ Process Driver Connection that switches the current process value of the Slider to the U16i_ex_alg_t1 tag. This Slider simulates the temperature value in the first container. In the same manner, configure a Slider for the remaining tags. To match the slider position with the current tag value upon opening the picture, a C-Action is created at Events ➔ Miscellaneous ➔ Open Picture that performs this task.</td>
</tr>
<tr>
<td>2</td>
<td>in addition, a Smart Object ➔ I/O Field is assigned to each Slider to display the current tag value. In the sample, these are the I/O Field1 to I/O Field6 objects. For I/O Field1, create Tag Connection at Properties ➔ Miscellaneous ➔ Picture Name to the U16i_ex_alg_t1 tag and trigger it upon change. This is the I/O Field assigned to the first Slider. In the same manner, assign an I/O Field to each remaining Slider.</td>
</tr>
<tr>
<td>3</td>
<td>The display of the individual containers is realized via the Smart Object Tank4 from the standard library. In this sample, these are the Tank41, Tank42 and Tank43 objects. These objects are only used for display purposes and receive no dynamics.</td>
</tr>
<tr>
<td>4</td>
<td>A Smart Object ➔ Status Display is assigned to each container that displays a warning light. In this sample, these are the Status Display1 to Status Display3 objects. In this sample, the bitmap Blinker blinkt nicht.gif is set as the Basic Picture and the bitmap Blinker blinkt.gif as the Flash Picture of Status Display1 at Properties ➔ State for the Current Status 0. The Property ➔ State ➔ Flashing Flash Picture Active is set to no. For the same property, a C-Action is created that activates the flashing if an alarm message for the corresponding container is pending. The other two Status Displays are configured in the same manner.</td>
</tr>
</tbody>
</table>
### Step 5
An additional *Smart Object* → *Status Display* is configured that displays a horn. In the sample, this is the *Status Display* object. The bitmap *Hupe hupt nicht.gif* is set as the *Basic Picture* and the bitmap *Hupe hupt.gif* as the *Flash Picture* of this object at *Properties* → *State* for the *Current Status 0*. The *Property* → *State* → *Flashing Flash Picture Active* is set to *no*. At the same property, create a *C-Action* that activates the flashing if an alarm message pertaining to one of the three containers comes in, i.e. if the tag set in *Alarm Logging* for the *Container Error* message class, controlling the central indicator, takes on the status *1*. In the sample, this is the *U16i_ex_alg_10* tag.

At *Properties* → *Geometry* → *Width*, a *C-Action* is created that emits audible signals if the object is flashing.

![Image](image1.png)

![Image](image2.png)

### Step 6
To display the messages configured in *Alarm Logging*, a *WinCC Alarm Control* is used. It is selected from the Object Palette’s *Control* selection menu and then placed in the picture.

### Step 7
After placing the Control in the picture, its *configuration dialog* will be displayed automatically.

As the *Window Title*, *MessageWindow_01* is entered. The *Display* check-box remains deselected. In the *C-Actions* created later, this window title is used to reference the corresponding Control.

The *Toolbar* and *Status Bar* check-boxes are deselected. The *configuration dialog* can be exited by clicking on *OK*.

![Windows quick configuration properties](image3.png)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Open the Control’s Properties dialog. This dialog is displayed via a ( \text{ } ) on the Control. In the General Information tab, the selection button is used to match the background color to the existing project color scheme. The Selection button is used to select the single messages, previously created in Alarm Logging, to be displayed by the Control.</td>
</tr>
</tbody>
</table>

![WinCC Alarm Control Properties](image)

Window Title: [ ] Display

MessageWindow_01

[ ] Sizeable

Server Selection:

[ ] Selection...

[ ] All Servers

Background Color

[ ] Filter...

Window Type

- [ ] Message Window
- [ ] Short-Term Archive Window
- [ ] Long-Term Archive Window

[ ] OK
[ ] Cancel
[ ] Apply
[ ] Help
Step | Procedure: Implementation in the Graphics Designer
---|---
9 | Via a graph on the System Block Message Class → Container Error, 2 checkboxes are displayed in the right window. The checkboxes Container Alarm and Container Warning are selected. This means that in runtime only messages of the message class container error will be displayed in the message window.

In the Parameters tab, the checkboxes Line Title, Column Title and Change Column Width are deselected. In the Selection field, the Cell radio-button is selected.
Step | Procedure: Implementation in the Graphics Designer
--- | ---
11 | In the **Message Blocks** tab, the columns are selected that will later be displayed in the message line. In this sample, **system blocks** are selected in the **Type** field using the [ ] . In the right window, **Date**, **Time** and **Number** are selected. For the **User Text Blocks** entry, **Message Text** and **Point of Error** are selected. For the **Process Value Blocks** entry, **Value** is selected.

![WinCC Alarm Control Properties](image)

- **Selection**
  - **System blocks**
  - **User text block**
  - **Process value blocks**

- **Message Blocks**
  - **Selection**
    - **Message text**
    - **Point of Error**
    - **Source of Error**
    - **Help text**
    - **Block: 4**
    - **Block: 5**
    - **Block: 6**
    - **Block: 7**

- **Apply Project Settings**
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>In the <em>Message Line</em> tab, the previously selected <em>Message Blocks</em> are assigned to the message line. The <em>Available Message Blocks</em> fields lists the available columns. By pressing on the -&gt; button, each message block can individually be added to the message line. Via the &gt;&gt; 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 OK.</td>
</tr>
</tbody>
</table>

![WinCC Alarm Control Properties](image)

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>For the toolbar, several <em>Windows Objects</em> <strong>→</strong> <em>Buttons</em> are configured that simulate the pressing of the individual buttons via special standard functions.</td>
</tr>
</tbody>
</table>
| 14   | A *Button* 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:  

AXC_OnBtnSinglAckn(lpszPictureName, lpszObjectName)  
AXC_OnBtnHornAckn(lpszPictureName, lpszObjectName)  

| 15   | Additional buttons are configured. One *Button* for the group acknowledgment and one *Button* for calling the Infotext dialog. The associated standard functions are:  

AXC_OnBtnVisibleAckn(lpszPictureName, lpszObjectName)  
AXC_OnBtnInfo(lpszPictureName, lpszObjectName)  


### Step 16: Procedure: Implementation in the Graphics Designer

As the replacement for the button that turns the auto-scroll function on and off, a Smart Object ➔ Status Display is used. In this sample, this is the Status Display6 object.

At Properties ➔ State ➔ Current Status, a Tag Connection to the BINi_ex_alg_00 tag is created. This tag contains the information whether auto-scrolling is turned on or off. At Events ➔ Mouse ➔ Press Left, a C-Action is created that inverts the status of the BINi_ex_alg_00 tag and calls the standard function AXC_OnBtnScroll(lpszPictureName,lpszObjectName). At the opening of the picture, the BINi_ex_alg_00 tag is set to 0, since auto-scroll is turned if a message window is reselected.

---

#### C-Action at Status Display1

```c
#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

```c
#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

```c
#include "spdefap.h"

long main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    #pragma code ("winmm.dll")
    BOOL PlaySound(LPCSTR pczSound, HMODULE hMod, DWORD dwSound);
    #define SND_FILENAME 0x00000000L
    #define SND_ASYNC 0x00001
    #pragma code ()

    char szProjectName[MAX_PATH];
    CNN_ERROR Error;
    char szSoundFilePath[MAX_PATH] = "";
    char szSoundFile[MAX_PATH] = "Hupe.wav";

    if (GetFlashPicture(lpszPictureName, lpszObjectName)) {
        if (DMGetRuntimeProject(szProjectName, MAX_PATH, &Error)) {
            strcat(szSoundFilePath, szProjectName);
            strcat(szSoundFilePath, szSoundFile);
            PlaySound(szSoundFilePath, NULL, SND_FILENAME | SND_ASYNC);
        }
    }

    return 55;
}
```

- 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation of the Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Alarm Logging editor from the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>For each single message, Loop in Alarm 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, OpenPicture 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 Alarm Logging. In this sample, the ALGLoopInAlarm function has been created in the Global Script editor.</td>
</tr>
<tr>
<td>3</td>
<td>In the table window of Alarm Logging, the Loop in Alarm column to open the Loop in Alarm dialog of the selected single message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgement bit</th>
<th>Loop in Alarm</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Not set</td>
<td>0</td>
</tr>
</tbody>
</table>

| 4    | As the Function Name, the function ALGLoopInAlarm is used. For messages that refer to the motors of the first sample, the picture ex_3_chapter_02.pdl is used as the Picture Name/Call Parameter, for messages of the second sample, the picture ex_3_chapter_02a.pdl is used. |
Step | Procedure: Implementation of the Sample
--- | ---

<table>
<thead>
<tr>
<th>Loop In Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can use Loop In Alarm to select a picture that shows the part of the process in which the message occurred.</td>
</tr>
<tr>
<td>To do this, you normally use the &quot;Open Picture&quot; function.</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td><strong>Picture Name / Parameter:</strong> ex_3_chapter_02.pdl</td>
</tr>
<tr>
<td><strong>Function Name:</strong> ALGLoopInAlarm</td>
</tr>
</tbody>
</table>

5  | The configuration of the Loop In Alarm function can also be performed in the Tag/Action tab of the single message’s properties dialog at the Loop in Alarm field.  
| The configurations made in Alarm Logging are saved.

Implementation in the Graphics Designer

Step | Procedure:
--- | ---

1  | Open the Graphics Designer and create a new picture. In this sample, this is the ex_3_chapter_02b.pdl picture. |

2  | To display the messages configured in Alarm Logging, a WinCC Alarm Control is used. It is selected from the Object Palette’s Control selection menu and then placed in the picture. |

3  | After placing the Control in the picture, its configuration dialog will be displayed automatically.  
| The configuration dialog can be exited by clicking on OK.  
| Open the Control’s Properties dialog. This dialog is displayed via a D on the Control.  
<p>| In the General Information tab, all settings can be made. It is not necessary to make a selection, since all single messages appearing are to be displayed. |</p>
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>In the <em>Toolbar</em> tab, the following check-boxes are selected:</td>
</tr>
<tr>
<td></td>
<td>- Single Acknowledgment</td>
</tr>
<tr>
<td></td>
<td>- Group Acknowledgment</td>
</tr>
<tr>
<td></td>
<td>- Auto-Scroll On/Off</td>
</tr>
<tr>
<td></td>
<td>- Report Functions</td>
</tr>
<tr>
<td></td>
<td>- Beginning of the List</td>
</tr>
<tr>
<td></td>
<td>- End of List</td>
</tr>
<tr>
<td></td>
<td>- Next Message</td>
</tr>
<tr>
<td></td>
<td>- Previous Message</td>
</tr>
<tr>
<td></td>
<td>- Infotext</td>
</tr>
<tr>
<td></td>
<td>- Loop in Alarm</td>
</tr>
</tbody>
</table>

![WinCC Alarm Control Properties](image)
### Step 5

**Procedure:**

In the *Message Blocks* tab, the columns are selected that will later be displayed in the message line. In this sample, *system blocks* are selected in the *Type* field using the check box. In the right window, *Date*, *Time* and *Number* are selected. For the *User Text Blocks* entry, *Message Text* and *Point of Error* are selected.

![WinCC Alarm Control Properties](image1)

### Step 6

**Procedure:**

In the *Message Line* tab, the previously selected *Message Blocks* are assigned to the message line. The *Available Message Blocks* 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 OK.

![WinCC Alarm Control Properties](image2)
Project Function ALGLoopInAlarm

```c
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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of the Required Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A total of three tags of the Binary Tag type are created. In this sample, these are the BINi_ex_alg_00, BINi_ex_alg_01 and BINi_ex_alg_02 tags.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in Alarm Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Alarm Logging editor from the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>Via a ➔ on the Archives entry, the Archive Parameters dialog is opened.</td>
</tr>
</tbody>
</table>
### Step 3
In this dialog, the short-term archive check-box is selected.

![Archive Parameter Assignment](image1)

### Step 4
In the right window, the icon for the short-term archive will be displayed. Via a right-click on this icon, the properties dialog of the short-term archive is opened.

### Step 5
The archive is to be stored on disk. In the *Number of Entries* entry field, specify 200. A *Selection* is not performed.

![Archives](image2)
Implementation in the Graphics Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Graphics Designer and create a new picture. In this sample, this is the <code>ex_3_chapter_02c.pdl</code> picture.</td>
</tr>
<tr>
<td>2</td>
<td>After placing the Control in the picture, its configuration dialog will be displayed automatically. As the Window Title, <code>MessageWindow_04</code> is entered. The Display check-box remains deselected. In the C-Actions created later, this window title is used to reference the corresponding Control. The Toolbar and Status Bar check-boxes are deselected. The configuration dialog can be exited by clicking on OK.</td>
</tr>
<tr>
<td>3</td>
<td>Open the Control’s Properties dialog. This dialog is displayed via a button on the Control. In the General Information tab, all settings can be made. It is not necessary to make a selection, since all single messages appearing are to be displayed.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Implementation in the Graphics Designer</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>In the <em>Message Blocks</em> tab, the columns are selected that will later be displayed in the message line. In this sample, <em>system blocks</em> are selected in the <em>Type</em> field using the <code>ckeck box</code>. In the right window, <em>Date</em>, <em>Time</em> and <em>Number</em> are selected. For the <em>User Text Blocks</em> entry, <em>Message Text</em> and <em>Point of Error</em> are selected.</td>
</tr>
</tbody>
</table>

![WinCC Alarm Control Properties](image_url)
### Step 5
In the *Message Line* tab, the previously selected *Message Blocks* are assigned to the message line. The *Available Message Blocks* 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 **OK**.

![WinCC Alarm Control Properties](image)

### Step 6
For the toolbar, several *Windows Objects* ⇒ *Buttons* are configured that simulate the pressing of the individual buttons via special standard functions.

### Step 7
A *Button* for calling the selection dialog and a *Button* for calling the Infotext dialog are configured. The associated standard functions are:

\[
\text{ACX_OnBtnInfo()ACX_OnBtnSelect()}
\]

### Step 8
As the replacement for the button that turns the auto-scroll function on and off, a *Smart Object* ⇒ *Status Display* is used. In this sample, the *Status Display3* object is used.

At Properties ⇒ State ⇒ Current Status, a *Tag Connection* to the *BINi_ex_alg_00* tag is created. This tag contains the information whether auto-scrolling is turned on or off. At Events ⇒ Mouse ⇒ Press Left, a C-Action is created that inverts the status of the *BINi_ex_alg_00* tag and calls the standard function *ACX_OnBtnScroll()*. At the opening of the picture, the *BINi_ex_alg_00* tag is set to 0, since auto-scroll is turned if the message window is reselected.
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:

\[ ACX\_OnBtnMsgFirst() \]
\[ ACX\_OnBtnMsgLast() \]
\[ ACX\_OnBtnMsgNext() \]
\[ ACX\_OnBtnMsgPrev() \]

These buttons are made inoperational via a Smart Object ➔ Graphic Object which is placed over them if auto-scroll is turned on. This is accomplished via a Tag Connection to the BINi\_ex\_alg\_00 tag at Properties ➔ Miscellaneous ➔ Display.

Via two Smart Objects ➔ Status Displays, the switching between the display types Message Window and Short-Term Archive Window is to be realized. The current status of the message window is stored in the BINi\_ex\_alg\_01 tag that must be set to zero at the opening of the picture, since the message window is displayed as a Short-Term Archive Window when reopened.

For Status Display1, create a Tag Connection at Properties ➔ State ➔ Current Status to the tag BINi\_ex\_alg\_01. At Properties ➔ Miscellaneous ➔ Operator-Control Enable, create a Dynamic Dialog that makes the object operational only if the message window displays the short-term archive, i.e. the BINi\_ex\_alg\_01 tag has the status 0. At Events ➔ Mouse ➔ Press Left, create a C-Action that simulates the pressing of the corresponding button on the toolbar and inverts the BINi\_ex\_alg\_01 tag. The Status Display2 object is configured in the same way. The following standard functions are used:

\[ ACX\_OnBtnMsgWin() \]
\[ ACX\_OnBtnArcShortt() \]

Via two additional Windows Objects ➔ Buttons, 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 Global Script editor. The message numbers, in between which the displayed messages are lying, are transferred to this function. In the sample, this function is called SetMsgNrSelection.
Project Function for Setting a Selection

```
BOOL SetMsgWinSelection(DWORD dwFrom, DWORD dwTo, LPSTR MsgTen)
{
    PCNH_ERROR pError;
    BOOL fRet;
    MSG_FILTER_STRUCT Filter;

    nmsgset(&Filter, 0, sizeof(MSG_FILTER_STRUCT));
    strcpy(Filter.szFilterName, MsgTen);
    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;
    }

    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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of the Required Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a total of three tags of the Unsigned 16-Bit Value type in Tag Management. In this sample, these are the U16i_ex_alg_20, U16i_ex_alg_21 and U16i_ex_alg_22 tags. They serve as status, lock and acknowledge tags.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of new Group Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Alarm Logging editor.</td>
</tr>
<tr>
<td></td>
<td>- On the Group Messages entry to expand it - two sub-entries will be displayed. These are the Message Class and User-Defined points.</td>
</tr>
<tr>
<td></td>
<td>- Via a click on the User-Defined entry, the New Group Message dialog is accessed.</td>
</tr>
<tr>
<td>2</td>
<td>In the displayed dialog, Alarm Container is entered as the Name. As the Status, Lock and Acknowledge Tags, the previously created tags are set. As the bit number, 0 is used in each case.</td>
</tr>
<tr>
<td></td>
<td>Exit the dialog box by clicking on OK.</td>
</tr>
<tr>
<td>3</td>
<td>In the same manner, two additional group messages are created. They use the same Status, Lock and Acknowledge Tags, but the bit numbers 1 and 2 respectively. In the right window, the icons of the newly created group messages are displayed.</td>
</tr>
</tbody>
</table>

![Properties dialog](image)
### Step 4

**Procedure: Creation of new Group Messages**

4. Via a [R] on one of these icons, the *New Single Message(s)* 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 *OK*.

#### New Single Message(s)

<table>
<thead>
<tr>
<th>Message Number(s)</th>
<th>Separate single messages by commas or indicate a range</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:18</td>
<td></td>
</tr>
</tbody>
</table>

- Only then insert the single message(s), if it does not already belong to a group.

5. Via a [D] on the *User-Defined* entry in the navigation window, the individual group messages are displayed as sub-entries. If one of these entries is selected with the [R], the right window will display the icons of all added single messages.

   - Group messages
   - Message Class
   - User defined
     - Alarm Container
     - Failure Motor
     - Warning Container

---

### Implementation in the Graphics Designer

**Step 1**

**Procedure: Implementation in the Graphics Designer**

1. In the *Graphics Designer*, a new picture is created. In this sample, this is the *ex_8_generator_00* picture.

   In this picture, a *Smart Object* ➔ *Status Display* is configured, which displays the current status of the group messages. In the sample, this is the *Status Display1* object. Following the configuration, the status of the group messages is stored by *Alarm Logging* in the *U16i_ex_alg_20* tag.

   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 *Properties ➔ State ➔ Current Status*, a *C-Action* is created that controls the status depending on the *U16i_ex_alg_20* tag and the required priority.
Step | Procedure: Implementation in the Graphics Designer
--- | ---
2 | Additionally, a Windows Object Button 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 Button1 object.
   Via a C-Action at Events Mouse Mouse Action, the current status of the group message is requested and the corresponding picture change performed.
   If no message is pending, an additional Windows Object Button is placed over the just described Button. This button makes the other one inoperational and displays this visually. At Properties Miscellaneous Operator-Control Enable, set this button to No.

3 | Configure another Windows Object Button which is used to acknowledge the currently displayed group message. In the sample, this is the Button3 object.
   Via a C-Action at Events Mouse Mouse Action, 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 U16i_ex_alg_22 is set and then immediately reset. If no unacknowledged message is pending, an additional Windows Object Button is placed over the just described Button to make it inoperational and to display this visually. At Properties Miscellaneous Operator-Control Enable, set this button to No.

4 | Configure another picture; in the sample, this is the ex_8_generator_01 picture.
   In this picture, three Windows Objects Check-Boxes are configured. In the sample, these are the Check-Box1, Check-Box2 and Check-Box3 objects.
   At Events Property Topics Output/Input Selected Boxes, create a C-Action for each check-box that locks or enables its corresponding group message. The respective locks are stored by Alarm Logging in the U16i_ex_alg_21 tag according to the configuration. Since a lock can also be set from the other side, a C-Action must be created at Properties Output/Input Selected Boxes. This action is triggered upon the change of the U16i_ex_alg_21 tag and checks if the status of the lock controlled by the corresponding Check-Box has changed.
### Step 5

In the initially created picture `ex_8_generator_00`, a Smart Object → Picture Window is created in which the `ex_8_generator_01` picture is set at Properties → Miscellaneous → Picture Name. Set the Property → Miscellaneous → Display to No.

Another Windows Object → Button is required that makes the previously configured Picture Window visible via a Direct Connection at Events → Mouse → Mouse Action.

---

### C-Action for Determining the current Status

```c
#include "allocated.h"

long main(char* lpszObjectName, char* lpszObjectModuleName, char* lpszPropertyName)
{
    WORD state;

    state = GetTagWord("UL61_ex_alg_20");

    if ((state&1) & (state&256)) return 6;
    else if ((state&2) & (state&512)) return 5;
    else if (state&3) return 3;
    else if ((state&4) & (state&1024)) return 4;
    else if (state&62) return 2;
    else if (state&64) 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

```c
#include "opdtop.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{  
  int value;
  Value = GetIndex(lpszPictureName, "Status Display");
  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

```c
#include "opdtop.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{  
  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&64)&&(state&512))
  SetTagWord("U16i_ex_alg_23", 
          (WORD)(2&GetTagWord("U16i_ex_alg_22")));
  SetTagWord("U16i_ex_alg_23", 
          (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

```c
#include <apdssig.h>
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* lpszObjectPathName, void* pvObjectInfo)
{
    DWORD dwServiceID;
    MSG_RTGRPSET_STRUCT mGroup;
    CRC_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);

    mGroup.fIDUsed = FALSE;
    strncpy(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():\nError.szErrorText = \n\n".szError.szErrorText);
    else
        printf("Executed MSRTLockGroup():\n\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

In runtime, the samples pertaining to this topic are accessed by selecting the button displayed above using the \( \text{\#} \). 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 @PDL picture (compact).rpl. This layout is copied and adapted to your needs.

Implementation in the Report Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Report Designer editor from the WinCC Explorer.</td>
</tr>
<tr>
<td></td>
<td>Report Designer</td>
</tr>
<tr>
<td></td>
<td>Layout</td>
</tr>
<tr>
<td></td>
<td>Line pr</td>
</tr>
<tr>
<td></td>
<td>Print jo</td>
</tr>
<tr>
<td></td>
<td>Open</td>
</tr>
<tr>
<td></td>
<td>New layout</td>
</tr>
<tr>
<td></td>
<td>Find...</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td>2</td>
<td>Via the File → Open... menus, open the system layout @PDLPic.rpl and save it under a different name via the File → Save As... menus. In this sample, the name PDLpicture.rpl is used.</td>
</tr>
</tbody>
</table>
3 Via a \textbf{R} on the blank space of the layout, its properties dialog is opened. In the \textit{Properties} tab, the general geometric settings can be made at the \textit{Geometry} property.

At the \textit{Miscellaneous} property, you can specify a cover sheet and a final page for the report. For this sample, a cover sheet is specified.

<table>
<thead>
<tr>
<th>Layout</th>
<th>Attribute</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colors</td>
<td>Width</td>
<td>21.00 cm</td>
</tr>
<tr>
<td>Geometry</td>
<td>Height</td>
<td>29.70 cm</td>
</tr>
<tr>
<td>Paper size</td>
<td>Paper size</td>
<td>A4 Sheet, 210- by 297-mm</td>
</tr>
<tr>
<td>Orientation</td>
<td>Orientation</td>
<td>Portrait</td>
</tr>
<tr>
<td>Left print margin</td>
<td>Left print margin</td>
<td>2.00 cm</td>
</tr>
<tr>
<td>Right print margin</td>
<td>Right print margin</td>
<td>2.00 cm</td>
</tr>
<tr>
<td>Top print margin</td>
<td>Top print margin</td>
<td>1.50 cm</td>
</tr>
<tr>
<td>Bottom print margin</td>
<td>Bottom print margin</td>
<td>1.50 cm</td>
</tr>
<tr>
<td>Left dynamic margin</td>
<td>Left dynamic margin</td>
<td>2.00 cm</td>
</tr>
<tr>
<td>Right dynamic margin</td>
<td>Right dynamic margin</td>
<td>2.00 cm</td>
</tr>
<tr>
<td>Top dynamic margin</td>
<td>Top dynamic margin</td>
<td>4.00 cm</td>
</tr>
<tr>
<td>Bottom dynamic margin</td>
<td>Bottom dynamic margin</td>
<td>4.00 cm</td>
</tr>
</tbody>
</table>

4 Via the toolbar buttons displayed below, the static and dynamic parts of the report can be edited.

Static Part Dynamic Part
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The dynamic part of the report contains a <em>Dynamic Object ➔ Embedded Layout</em>. In the sample, this is the <em>EmbedLayout</em> object, which must be changed. Via a <img src="-image" alt="Update Symbol" /> on the dynamic part of the report, select the <em>@PDL picture (compact).rpl</em> layout at <strong>Properties ➔ Miscellaneous ➔ Layout File</strong>. 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 <em>EmbedLayout</em> object via <strong>Properties ➔ Miscellaneous ➔ Layout File</strong>.</td>
</tr>
</tbody>
</table>

![Object Properties](image)
### Step 6

**Procedure: Implementation in the Report Designer**

6. The static part of the report contains a header and a footer.

   The footer contains the **System Objects Date/Time, Page Number, Project Name** and **Layout Name**.

   The header contains two **Static Objects** ➔ **Static Texts** as well as a **System Object** ➔ **Project Name**. Additionally, a logo is displayed using a **Static Object** ➔ **OLE Element**. In this sample, the text of the StatText1 object is changed to **Picture Documentation**. To display your own logo, the existing **OLEElement1** object is deleted. A new **Static Object** ➔ **OLE Element** is now configured. From the **Insert Object** dialog, which is displayed after placing the object in the report, select the **Create from File** radio-button and specify the bitmap file that contains the logo. The dialog is closed by clicking on **OK**.

### Insert Object

![Insert Object Dialog]

- **Create New**
- **Create from File**

File: `\ZIP-SERVER\Projekthandbuch\Projekt\`

- **Browse**
- **Link**
- **Display As Icon**

**Result**

Inserts the contents of the file as an object into your document so that you may activate it using the program which created it.

### Step 7

7. Via the toolbar buttons displayed below, you can switch among the cover sheet, report contents and final page.

   In this sample, the cover sheet contains two **Static Objects** ➔ **Static Texts**, a **System Object** ➔ **Project Name** and a **Static Object** ➔ **Static Metafile**.

### Step 8

8. The changes made in the Report Designer are saved and the Report Designer editor is exited.
Creation of a Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Print Job</th>
</tr>
</thead>
</table>
| 1    | In the *WinCC Explorer*, a new print job is created via a `R`.  
|      | ![Image of Report Designer]  
|      | ![Image of Print Job]  
|      | ![Image of Print Job Properties]  
|      | **Name:** PrintJob_pdl  
|      | **Project:** \ZIP:SERVER1\Projektierhandbuch\Project_  
|      | **Layout:** PDLpicture.rpl  
|      | ![Image of Line layout for line printer]  
|      | **Last printout at:**  
|      | **Next printout at:**  
|      | **Start Parameter**  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | **Cycle:** NONE  
|      | ![Image of OK, Cancel, Help buttons]  
| 2    | In the right window, this new print job, with the default name *Print Job001*, will be added to the existing print jobs. Via a `D` or a `R` on this print job, its properties dialog is opened.  
|      | ![Image of Print Job Properties]  
|      | **Name:** PrintJob_pdl  
|      | **Project:** \ZIP:SERVER1\Projektierhandbuch\Project_  
|      | **Layout:** PDLpicture.rpl  
|      | ![Image of Line layout for line printer]  
|      | **Last printout at:**  
|      | **Next printout at:**  
|      | **Start Parameter**  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | ![Image of Date and Time]  
|      | **Cycle:** NONE  
|      | ![Image of OK, Cancel, Help buttons]  

*WinCC Editors (Project_WinCCEditors)*
### Procedure: Creation of a Print Job

In the *Print Job* tab, the default name is replaced with *Printjob_pdl*. As the *Layout*, the previously created *PDLpicture.rpl* layout is specified.

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>In the <em>Print Job</em> tab, the default name is replaced with <em>Printjob_pdl</em>. As the <em>Layout</em>, the previously created <em>PDLpicture.rpl</em> layout is specified.</td>
</tr>
</tbody>
</table>

![Print Job Properties](image-url)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>In the <em>Selection</em> tab, you can specify what to print. In the <em>Page Range</em> field, the <em>All</em> radio-button is selected. A <em>Time Range</em> set will have no influence in this sample.</td>
</tr>
</tbody>
</table>

![Print Job Properties](image)

*Print Job Properties*

- **Page Range**
  - **All**
  - Pages from [ ] to [ ]

- **Time Range**
  - Relative (backwards from the print start time)
    - Number: [ ]
      - All [ ]
  - Absolute
    - From: [ ]
      - [ ]
    - To: [ ]
      - [ ]
### Step Procedure: Creation of a Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>In the <strong>Printer Setup</strong> tab, the printer to be used is specified. The data can also be printed to a file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><strong>Print Job Properties</strong>&lt;br&gt;Print output to&lt;br&gt;&lt;br&gt;Printer&lt;br&gt;&lt;br&gt;Printer Priorities&lt;br&gt;1.) &lt;Standard Printer&gt;&lt;br&gt;2.) &lt;NONE&gt;&lt;br&gt;3.) &lt;NONE&gt;&lt;br&gt;&lt;br&gt;File [*.emf]&lt;br&gt;&lt;br&gt;Tray: PRT_OUT_JJJJMMTTHHMM...&lt;br&gt;&lt;br&gt;OK Cancel Help</td>
</tr>
<tr>
<td>6</td>
<td>Via <strong>R</strong>, the print job can be started from the <strong>WinCC Explorer</strong>. In the same way, a print preview can be displayed.</td>
</tr>
<tr>
<td>7</td>
<td>In the sample project, a preview of the print job can be activated via a <strong>Windows Object Button</strong>. This is the Button13 object in the <em>ex_3_chapter_03.pdl</em> picture.&lt;br&gt;At <strong>Events Mouse Mouse Action</strong>, create a <strong>C-Action</strong> that starts a preview of the corresponding print job.</td>
</tr>
</tbody>
</table>
C-Action for Starting a Print Job

```c
#include "apdctop.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.pdl)

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a new layout via a Report Designer on the Page Layout entry in the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>In the right window, this new layout, with the default name NewRPL00.RPL, will be added to the existing page layouts. Via a Report Designer, this layout can be renamed using a more meaningful name. In this sample, the name WinCC Explorer.rpl is used.</td>
</tr>
<tr>
<td>3</td>
<td>Open the new layout in the Report Designer. In the properties dialog of the layout, which is accessed via a Report Designer on the blank space of the report page, at the Miscellaneous property, a Cover Sheet is specified. The remaining settings are kept.</td>
</tr>
<tr>
<td>4</td>
<td>In the static part of the report, various Static Objects and System Objects are configured for the header and the footer. The design of the cover sheet is only a suggestion for your own design.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Implementation in the Report Designer</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>5</td>
<td>In the dynamic part of the report, a <em>Dynamic Object</em> → <em>Dynamic Table</em> is configured. In the sample, this is the <em>DynTable1</em> object. After the object has been placed in the report, the <em>Connect</em> dialog will be displayed. From the <em>WinCC Explorer</em> folder, select the <em>Tag</em> entry. Close the dialog box by clicking on <em>OK</em>.</td>
</tr>
</tbody>
</table>

![Connect](image)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Actions in Grafexe</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Alarm Logging CS</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Alarm Logging RT (M)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Control Center CS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Global Script</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Graphics Designer</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Group displays</strong></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Control Center CS. Tag</strong></td>
</tr>
</tbody>
</table>

| 6 | In the *Connect* tab of the table’s properties dialog, several selection options are available. |

![Diagram](image)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Tag parameter selection</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Tag group selection</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Tag selection</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Format</strong></td>
</tr>
</tbody>
</table>

| 7 | Via a *D* 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. |
### Step 8

**Procedure: Implementation in the Report Designer**

In the tag parameters selection dialog, the check-boxes *Data type*, *Group*, *Parameters* and *External Tag Data* are selected. Additionally, the check-box *Compact* is selected. It has the effect that all tag data is displayed in one line.

![Control Center CS: Select tag parameters](image)

#### 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.

### Step 9

For the tag group selection, the groups *AlarmLogging1* and *AlarmLogging2* are selected in this sample. This selection is only possible, if the check-box *All Tag Groups* has not been selected.

To apply the selection of the tag groups, the check-box *All Tags* in the tag selection dialog must be deselected.

The settings made in the **Report Designer** are saved.

### 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a new layout via a R on the Page Layout entry in the WinCC Explorer. In the right window, this new layout, with the default name NewRPL00.RPL, will be added to the existing page layouts. Via a R on this name, rename it to tlg_cs.rpl.</td>
</tr>
<tr>
<td>2</td>
<td>Open the new layout in the Report Designer. In the properties dialog of the layout, which is accessed via a R on the blank space of the report page, at the Miscellaneous property, a Cover Sheet is specified. The remaining settings are kept.</td>
</tr>
<tr>
<td>3</td>
<td>In the static part of the report, various Static Objects and System Objects are configured for the header and the footer. The design of the cover sheet is only a suggestion for your own design.</td>
</tr>
</tbody>
</table>
### Step 4

**Procedure: Implementation in the Report Designer**

In the dynamic part of the report, a *Dynamic Object* → *Dynamic Table* is configured. In the sample, this is the *DynTable1* object. After the object has been placed in the report, the *Connect* dialog will be displayed. From the *Tag Logging CS* folder, select the *Process Value Archive* entry. Close the dialog box by clicking on *OK*.

![Connect]

5. In the *Connect* tab of the table’s properties dialog, several selection options are available.
   - Archive names
   - Process archive data

6. Via a *D* 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.

In the dialog for the selection of the desired archives, the check-box *All Archives* is selected. In the dialog for the selection of the desired archive data, all listed data selections are check-marked.

The settings made in the *Report Designer* are saved.

---

**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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a new layout via a ( R ) on the Page Layout entry in the WinCC Explorer. In the right window, a new layout is added to the existing ones. The default name of this layout is changed via a ( R ) to tlg_ZS_PA_00.rpl.</td>
</tr>
<tr>
<td>2</td>
<td>Open the new layout in the Report Designer. In the static part of the report, various Static Objects and System Objects are configured for the header and the footer.</td>
</tr>
<tr>
<td>3</td>
<td>In the dynamic part of the report, three Dynamic Objects ( \rightarrow ) Dynamic Metafiles are configured. In this sample, these are the objects DynMetafile1, DynMetafile2 and DynMetafile3. After an object has been placed in the report, the Connect dialog will be displayed. For all three objects, the Tag Trend entry is selected from the Tag Logging Runtime folder. The dialogs are closed by clicking on OK.</td>
</tr>
</tbody>
</table>
### Step 4: Procedure: Implementation in the Report Designer

In the **Connect** tab of the **Dynamic Metafile**’s properties dialog, several selection options are available.

- **Time range**
- **Tag Selection**
- **Format**

Via a screenshot 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 Time Selection** is not made.

In the dialog pertaining to the **Tag Selection**, the **Tag Selection for Reporting** dialog is opened via ![Edit](#).

#### Tag Logging Runtime: Tag selection for reporting

The settings made in the **Report Designer** are saved. Via ![Add](#) **Add**, the Archive Data Selection dialog is opened. The archive **ZS_ProcessValueArchive_00** and the tag **G64_ex_tlg_01** are selected. The dialogs can be closed with **OK**. In the same way, configure the remaining two tags for the other objects.

#### Archive Selection

The settings made in the **Report Designer** are saved. Via ![Add](#) **Add**, the Archive Data Selection dialog is opened. The archive **ZS_ProcessValueArchive_00** and the tag **G64_ex_tlg_01** are selected. The dialogs can be closed with **OK**. In the same way, configure the remaining two tags for the other objects.
Creation of a Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Print Job</th>
</tr>
</thead>
</table>
| 1    | In the *WinCC Explorer*, a new print job is created via a ![Report Designer](image) on the *Print Job* entry. <br> - ![Layout](image)  
- ![Line printer layout](image)  
- ![Print job](image)  
  ![New print job](image)  
  ![End...](image)  
  ![Properties](image)  <br>In the right window, this new print job, with the default name *Print Job001*, will be added to the existing print jobs. Via a ![Report Designer](image) or a ![Print Job](image) on this print job, its properties dialog is opened. <br>In the *Print Job* tab, the *Name Printjob_ZS_PA_00* is entered. As the *Layout*, the previously created *tlg_ZS_PA_00.rpl* layout is specified. <br>In the *Selection* tab, the print range is set. In the *Page Range* field, the *All* radio-button is selected. In the *Time Range* field, the *Absolute* radio-button is selected. A specific time range is not set, this will be done later in runtime. <br>In the *Printer Setup* tab, the printer to be used is specified. |
| 2    | In the sample project, a preview of the print job can be activated via a *Windows Object* → *Button*. This is the *Button13* object in the *ex_3_chapter_01a.pdl* picture. <br>At *Events* → *Mouse* → *Mouse Action*, create a *C-Action* that starts a preview of the corresponding print job. |
Step | Procedure: Creation of a Print Job
---|---
3 | To make a time selection, a settings dialog is needed. This dialog is configured as a separate picture - in the sample, this is the `ex_5_window_02.PDL` picture.
   
   In this picture, six *Smart Objects* → *I/O Fields* 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 *Unsigned 16-Bit Value* type is configured in Tag Management for each *I/O Field*. In this sample, these are the tags `U16i_ex_rep_f1` to `U16i_ex_rep_f6` for the start time and the tags `U16i_ex_rep_t1` to `U16i_ex_rep_t6` for the end time. For each *I/O Field*, a Tag Connection is created at *Properties* → *Output/Input* → *Output Value* to its corresponding tag.
   
   At *Properties* → *Output/Input* → *Output Format*, set the format to 099 for all *I/O Fields*, except for the *I/O Fields* containing the year. These fields use the output format 09999.

<table>
<thead>
<tr>
<th>Time Selection for Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
</tr>
<tr>
<td>00</td>
</tr>
</tbody>
</table>

4 | At *Events* → *Miscellaneous* → *Open Picture* of the `ex_5_window_02.PDL` picture, a *C-Action* 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.

5 | In the `ex_5_window_02.PDL` picture, two *Windows Objects* → *Buttons* are configured. In the sample, these are the *Button1* and *Button2* objects.
   
   *Button2* is the Cancel button. At *Events* → *Mouse* → *Mouse Action*, a *Direct Connection* is created, which switches the *Constant 0* to the *Display of the Current Window*. *Button1* is the OK button. It also receives a *Direct Connection* at *Events* → *Mouse* → *Mouse Action* for closing the window. At *Events* → *Mouse* → *Press left*, a *C-Action* 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 *Text Tag 16-Bit Character Set* type that has to be created in *Tag Management*. In the sample, this is the `T16i_ex_rep_00` tag.

6 | To display the created picture in the `ex_3_chapter_01a.PDL` picture, a *Smart Object* → *Picture Window* must be created. In the sample, this is the *Picture Window1* object. At *Properties* → *Miscellaneous* → *Picture Name*, set the previously created picture `ex_5_window_02.PDL`. Set the *Display property* to *No*.

7 | To make the *Picture Window* visible, an additional *Windows Object* → *Button* is required - in the sample, this is the *Button12* object. This object receives a *C-Action* at *Events* → *Mouse* → *Mouse Action*, which writes the name of the print job to be processed to the `T16i_ex_rep_00` tag and makes the *Picture Window1* object visible.
C-Action at the OK Button

```c
#include "apdfunc.h"
void CnLButtonDown(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;
    PCM_ERROR pError;
    HPROPERTY hProp;
    LPVOID ptr1, ptr2;
    DWORD typ;
    DWORD dwval;
    TCHAR propname1[200], propname2[200];
    TCHAR g_szProj[MAX_PATH+1];

    typ = V1_DATE;
    strcpy( propname1, 'ABSOLUTESELECTFROM' );
    strcpy( propname2, 'ABSOLUTESELECTINTO' );
    ptr1 = (LPVOID)&st1;
    ptr2 = (LPVOID)&st2;

    //-------set project path
    if( !MpGetRuntimeProject( g_szProj, MAX_PATH, pError )) {
        print("Error MpGetRuntimeProject(\n"");
        return FALSE;
    }

    //-------create property handle
    hProp = RpJCreatePropertyHandle( g_szProj, pError);
    if( !hProp ) {
        print("Error RpJCreatePropertyHandle(\n"");
        return FALSE;
    }

    //-------set job properties
    if ( !RpJSetJobProps( hProp, jobname, pError )) {
        print("Error RpJSetJobProps(\n"");
        RpJDeletePropertyHandle( hProp, pError);
        return FALSE;
    }

    //-------set property
    if ( !RpJSetProperty( hProp, propname1, ptr1,
        (VARTYPE) typ, 200, pError )) {
        print("Error RpJSetProperty(\n"");
        RpJDeletePropertyHandle( hProp, pError);
        return FALSE;
    }

    //-------save job properties
    if ( !RpJSaveJobProps( hProp, jobname, pError )) {
        print("Error RpJSaveJobProps(\n"");
        RpJDeletePropertyHandle( hProp, pError);
        return FALSE;
    }

    //-------set job properties
    if ( !RpJGetJobProps( hProp, jobname, pError )) {
        print("Error RpJGetJobProps(\n"");
        RpJDeletePropertyHandle( hProp, pError);
        return FALSE;
    }

    //-------set property
    if ( !RpJSetProperty( hProp, propname2, ptr2,
        (VARTYPE) typ, 200, pError )) {
        print("Error RpJSetProperty(\n"");
        RpJDeletePropertyHandle( hProp, pError);
        return FALSE;
    }

    //-------save job properties
    if ( !RpJSaveJobProps( hProp, jobname, pError )) {
        print("Error RpJSaveJobProps(\n"");
        RpJDeletePropertyHandle( hProp, pError);
        return FALSE;
    }

    //-------delete property handle
    fRet = RpJDeletePropertyHandle( hProp, pError);
    return fRet;
}
• 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a new layout via a ![R] on the Page Layout entry in the WinCC Explorer. In the right window, a new layout is added to the existing ones. The default name of this layout is changed via a ![R] to tlg_ZK_PA_02.rpl.</td>
</tr>
<tr>
<td>2</td>
<td>Open the new layout in the Report Designer. In the static part of the report, various Static Objects and System Objects are configured for the header and the footer.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Implementation in the Report Designer</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>In the dynamic part of the report, a Dynamic Object → Dynamic Table is configured. In the sample, this is the DynTable1 object. After the object has been placed in the report, the Connect dialog will be displayed. From the Tag Logging Runtime folder, the Tag Table entry is selected. Close the dialog box by clicking on OK.</td>
</tr>
</tbody>
</table>

![Connect dialog](image)

Tag Logging Runtime MultiClient.Tag Table
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>In the Connect tab of the Dynamic Metafile’s properties dialog, several selection options are available.</td>
</tr>
<tr>
<td></td>
<td>![Time range] ![Tag Selection]</td>
</tr>
<tr>
<td></td>
<td>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 Time Range is not specified.</td>
</tr>
<tr>
<td></td>
<td>Via [ ] ![Edit] ![Add], the dialog for the archive selection is archived. In this dialog, the ZK_ProcessValueArchive_00 archive and its archive tags are selected. The dialogs can be closed with OK.</td>
</tr>
</tbody>
</table>

The settings made in the **Report Designer** are saved.
Creation of a Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the WinCC Explorer, a new print job is created via a R.</td>
</tr>
<tr>
<td></td>
<td>In the right window, this new print job, with the default name Print Job001, will be added to the existing print jobs. Via a D or a R on this print job, its properties dialog is opened.</td>
</tr>
<tr>
<td></td>
<td>In the Print Job tab, the Name Printjob_ZK_PA_02 is entered. As the Layout, the previously created tlg_ZK_PA_02.rpl layout is specified.</td>
</tr>
<tr>
<td></td>
<td>In the Selection tab, the print range is set. In the Page Range field, the All radio-button is selected. In the Time Range field, the Absolute radio-button is selected. A specific time range is not set, this will be done later in runtime.</td>
</tr>
<tr>
<td></td>
<td>In the Printer Setup tab, the printer to be used is specified.</td>
</tr>
</tbody>
</table>

2 | The procedure for making a time selection in runtime is detailed in the previous sample at Creation of a Print Job. |
|   | Adaptations must be made for the button displayed below in the C-Action at Events → Mouse → Mouse Action. |
|   | The text tag T16i_ex_rep_00 must be supplied with the name of the just created print job. |
|   | For the button starting the print preview, the print job name in the function call of the C-Action at Events → Mouse → Mouse Action must also be changed. |

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the system layout @alrtmef.rpl in the WinCC Explorer via a <strong>R</strong> on its name.</td>
</tr>
<tr>
<td></td>
<td>Via the File → <strong>Save As...</strong> menus, the system layout is saved under a different name. In this sample, the name <strong>alg_mef.rpl</strong> is used.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Implementation in the Report Designer</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
</tbody>
</table>
| 3    | The dynamic part of the report consists of a *Dynamic Object* \(\Rightarrow\) *Dynamic Table*. In the sample, this is the *DynTable1* object.  
The properties dialog of the *DynTable1* object is opened and the *Connect* tab selected. The table is already connected to the *Message Sequence Report of Alarm Logging Runtime*. A selection has also been already performed. |
| ![Object Properties](image) |
| 4    | Via \(\Rightarrow\) *Edit* or a \(\Rightarrow\) *D* on the *Selection* entry, the dialog for the message block selection is opened. In this dialog, the system blocks *Date*, *Time*, *Number* and *Loop in Alarm* are already selected. In this sample, all remaining message blocks are selected via the button displayed below.  
The dialog is closed by clicking on *OK*. The layout is saved. |
| ![Selection Dialog](image) |
Adaptation of the Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Adaptation of the Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the system print job @Report Alarm Logging RT Message sequence via a D or a R on its name in the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>In the Print Job tab, the just created Layout altrmef.rpl is specified. In addition, the desired printer is set in the Printer Setup tab. No other adaptations are needed. Close the dialog box by clicking on OK.</td>
</tr>
<tr>
<td>3</td>
<td>The message sequence report must be activated in the Alarm Logging editor. To do this, open this editor. Via a R on the Reports entry, the Assigning Report Parameters dialog is opened.</td>
</tr>
<tr>
<td></td>
<td>![Report] ![Add/Remove... Properties]</td>
</tr>
<tr>
<td></td>
<td>In this dialog, the Check-Box for the message sequence report is selected.</td>
</tr>
<tr>
<td>4</td>
<td>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. 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 Report Functions. In the sample project, this function has already been selected in the MessageWindow_04 template used in the ex_3_chapter_02b.pdl picture.</td>
</tr>
<tr>
<td></td>
<td>![Report Functions]</td>
</tr>
<tr>
<td></td>
<td>The settings made in Alarm Logging are saved.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Line Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creation of a new line layout via a R on the appropriate entry in the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>A new line layout with the default name NewRP100.RP1 is created. This name is kept for the sample project. With a D on the name of the new line layout in the right window, the Line Layout Editor is opened. In this editor, general settings pertaining to page margins, headers, footers, etc. can be made. In the Table field, a dialog for the selection of the desired message blocks in the message sequence report is opened via Selection. In this sample, all available message blocks are selected.</td>
</tr>
<tr>
<td>3</td>
<td>The number of columns as well as their width is matched automatically to the selected message blocks and their order. The settings made are saved and the line layout editor closed.</td>
</tr>
</tbody>
</table>
Adaptation of the Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Adaptation of the Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the system print job @Report Alarm Logging RT Message sequence via a D or a R on its name in the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>In the Print Job tab, the Line Layout for Line Printer check-box is selected and the just created Layout NewRP100.RP1 specified. In addition, the desired line printer is set in the Printer Setup tab. No other adaptations are needed. Close the dialog box by clicking on OK.</td>
</tr>
</tbody>
</table>

3 The message sequence report must be activated in the Alarm Logging editor. To do this, open this editor. Via a R on the Reports entry, the Assigning Report Parameters dialog is opened. In this dialog, the Check-Box 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Report Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the system layout @alrtuma.rpl in the WinCC Explorer via a R on its name in the right window. Via the File Save As... menus, the system layout is saved under a different name. In this sample, the name alg Uma is used.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>The dynamic part of the report consists of a Dynamic Object Dynamic Table. In the sample, this is the DynTable1 object. The properties dialog of the DynTable1 object is opened and the Connect tab selected. The table is already connected to the Short-Term Archive Report of Alarm Logging Runtime. A selection has also been already performed.</td>
</tr>
<tr>
<td>4</td>
<td>Via the Edit button or a D on the Selection entry, the dialog for the message block selection is opened. In this dialog, the system blocks Date, Time and Number are already selected. In this sample, all remaining message blocks are selected via the button displayed below. The dialog is closed by clicking on OK. The layout is saved.</td>
</tr>
</tbody>
</table>

>>
Adaptation of the Print Job

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Adaptation of the Print Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the system print job @Report Alarm Logging RT Revolving archive via a D or a R on its name in the WinCC Explorer.</td>
</tr>
<tr>
<td>2</td>
<td>In the Print Job tab, the just created Layout alr_tuma.rpl is specified. In addition, the desired printer is set in the Printer Setup tab. No other adaptations are needed. Close the dialog box by clicking on OK.</td>
</tr>
<tr>
<td>3</td>
<td>The archive report must be activated in the Alarm Logging editor. To do this, open this editor. Via a R on the Reports entry, the Assigning Report Parameters dialog is opened. In this dialog, the Check-Box for the archive report is selected. The settings made in Alarm Logging are saved.</td>
</tr>
<tr>
<td>4</td>
<td>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 ACX_OnBtnPrint(). In the sample project, this button is already configured in the ex_3_chapter_02c.pdl picture.</td>
</tr>
</tbody>
</table>

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 OLE. 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the WinCC Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In Tag Management, several tags of various types are created. In the sample, the following tags were used:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Table" /></td>
</tr>
<tr>
<td></td>
<td><strong>Name</strong></td>
</tr>
<tr>
<td></td>
<td>BINl_excel_00</td>
</tr>
<tr>
<td></td>
<td>F32l_excel_00</td>
</tr>
<tr>
<td></td>
<td>F64l_excel_00</td>
</tr>
<tr>
<td></td>
<td>S16l_excel_00</td>
</tr>
<tr>
<td></td>
<td>S32l_excel_00</td>
</tr>
<tr>
<td></td>
<td>S64l_excel_00</td>
</tr>
<tr>
<td></td>
<td>U16l_excel_00</td>
</tr>
<tr>
<td></td>
<td>U32l_excel_00</td>
</tr>
<tr>
<td></td>
<td>U64l_excel_00</td>
</tr>
<tr>
<td>2</td>
<td>In a picture, a Smart Object -&gt; I/O Field is created for each tag. At Properties -&gt; Output/Input -&gt; Output Value, a Tag Connection is created to each tag.</td>
</tr>
</tbody>
</table>
Implementation in Excel (Version 8.0)

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in Excel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a new Excel folder. In this sample, the folder is named <em>OLE_Communication.xls</em>. In a spreadsheet, fill out a column with the names of the tags to be processed.</td>
</tr>
<tr>
<td>2</td>
<td>Configure a button for reading the tag values. For this purpose, activate the <em>Control Toolbox</em> toolbar via the <em>View</em> ➔ <em>Toolbars</em> menus, if it has not be activated yet. Select the Command Button control element and place it in the table.</td>
</tr>
</tbody>
</table>
| 3    | 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 *GetValue* and as the caption, *GetValue()*.

![Control Toolbox](image1.png)  
![GetValue](image2.png)  

| 4    | With a on the button created, the Visual Basic Editor is opened and the procedure to be executed can be entered. |
| 5    | As described in step 2, configure another button for writing tag values. |

![Excel Spreadsheet](image3.png)
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
        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 bSet 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
        bSet = 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 set 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 to 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Two tags of the Binary Tag type are used. These are the BINi_ex_org_00 and BINi_ex_org_01 tags. These tags make the next and back buttons operational. Additionally, a tag of the Unsigned 16-Bit Value type is used. In the sample, this is the U16i_ex_org_00 tag. This tag stores the current position in the picture index. Also, a tag of the Text Tag 16-Bit Character Set type is used. In the sample, this is the T16x_ex_org_00 tag. This tag records the current picture name.</td>
</tr>
<tr>
<td>2</td>
<td>A project function for controlling the picture index is available. This is the CreatePictureSequence function. This function is called every time a picture change is performed. This is achieved by a C-Action at Events ➔ Property Topics ➔ Miscellaneous ➔ Picture Name of the workspace object in the ex_0_startpicture_00.pdl 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.</td>
</tr>
<tr>
<td>3</td>
<td>In the ex_2_keyboard_00.pdl keyboard picture, two control elements for moving backwards and forwards are configured. If one of these objects is pressed, the project function CreatePictureSequence is called as well. This function will then perform the picture change. Via two Smart Objects ➔ Graphic Objects, these two control elements can be made inoperational.</td>
</tr>
<tr>
<td>4</td>
<td>Via the following button, a picture is called that displays the current picture index. In the sample, this is the ex_9_register_00.PDL picture.</td>
</tr>
<tr>
<td>5</td>
<td>In this picture, 10 Standard Objects ➔ Static Texts are configured. The</td>
</tr>
</tbody>
</table>
### Step | Procedure: Implementation in the Graphics Designer
--- | ---
 | default texts are deleted from all objects. At the picture object’s Properties ➔ Geometry ➔ Picture Width, a C-Action is created that calls the project function CreatePictureSequence. This action provides the Static Texts with the stored picture names. This C-Action is triggered upon the change of the tag T16x_ex_org_00. This results in the update of the display upon a picture change. For each Static Text, a C-Action is created at Events ➔ Mouse ➔ Press left 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 Dynamic Dialog at Properties ➔ Colors ➔ Font Color for each Static Text. Via a parameter, the project function recognizes from which position it was called. For this parameter, various constants are defined in the APDEFAP.H file. This file is located in the Library subfolder of the project folder. The following constants were defined:
  * #define REG_INSERTPICTURE 0
  * #define REG_BACK 1
  * #define REG_FORWARD 2
  * #define REG_DIRECT 3
  * #define REG_SHOWREGISTER 4

6 | ex_3_chapter_02pdl
  | ex_3_chapter_02.PDL
  | ex_3_chapter_01f.PDL
  | ex_3_chapter_01e.PDL
  | ex_3_chapter_01d.PDL
  | ex_3_chapter_01c.PDL
  | ex_3_chapter_01b.PDL
  | ex_3_chapter_01a.PDL
  | ex_3_chapter_01.PDL
  | ex_3_chapter_00.PDL
Project Function for Controlling the Picture Index

```c
#include "APDEHAP.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){
        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_SHOUREGISTER){
        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 )
    {
        if (pos<(biz-1))
            SetTagBit("BIN_ex_ort_00",FALSE);
        else
            SetTagBit("BIN_ex_ort_00",TRUE);

        if (pos>0)
            SetTagBit("BIN_ex_ort_01",FALSE);
        else
            SetTagBit("BIN_ex_ort_01",TRUE);
    }

    SetTagWord("U16i_ex_ort_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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Two tags of the <em>Unsigned 16-Bit Value</em> type are used. In the sample, these are the <em>U16i_ex_con_00</em> and <em>U16i_ex_con_01</em> tags. These tags store the currently selected chapter number and the sample number.</td>
</tr>
<tr>
<td>2</td>
<td>A picture is available, in which the layout of the dialog has been created. This is the <em>ex_9_register_01.pdl</em> picture. For each chapter, a <em>Standard Object</em> ➔ <em>Static Text</em> and a <em>Smart Object</em> ➔ <em>Graphic Object</em> is configured. Initially, at the selection of the picture, no chapter is selected - the <em>U16i_ex_con_00</em> tag has the value zero. If a <em>Static Text</em> is selected with the , the corresponding chapter number is written to this tag. Using various <em>Dynamic Dialogs</em>, the coloring of the selected <em>Static Text</em> is changed.</td>
</tr>
</tbody>
</table>
### Step Procedure: Implementation in the Graphics Designer

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>For each chapter, a separate picture is configured, which - depending on the chapter number selected - is displayed in a <strong>Smart Object</strong> → <strong>Picture Window</strong>. If no chapter is selected, the picture window is not displayed. For each sample (of the chapters), a <strong>Standard Object</strong> → <strong>Static Text</strong> is configured. If a chapter is selected, no sample will be displayed initially. If a <strong>Static Text</strong> is selected with the √, the corresponding sample number is written to the U16i_ex_con_01 tag. Using various <strong>Dynamic Dialogs</strong>, the coloring of the selected <strong>Static Text</strong> is changed.</td>
</tr>
<tr>
<td>4</td>
<td>To implement the picture selection via a double-click, three external C variables (listed below) were created. They are created in the project function <strong>CreateExternal</strong>. This function is performed once at the start of the project. • extern BOOL bPress1, bPress2 • extern int nButtonID Via a <strong>C-Action</strong> at <strong>Properties</strong> → <strong>Geometry</strong> → <strong>Picture Width</strong> 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 <strong>nButtonID</strong> variable.</td>
</tr>
</tbody>
</table>

#### C-Action at a Sample Text

```c
#include "apdsap.h"
void OnButtonDown(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("Ul6i_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

```c
#include "apdelet.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_01h.PDL");
    break;
  case10 SetPictureName("ex_0_startpicture_00.PDL", "workspace","ex_3_chapter_01i.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 D 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Five tags of the Unsigned 32-Bit Value type are used. In the sample, these are the CO_TIME, CO_MAX, CO_MIN, CO_MEAN and CO_BACK tags. These tags store the current color values. An additional tag of the Unsigned 32-Bit Value type is used. In the sample, this is the CO_TEMP tag. This tag serves as a buffer for a color value that is to be applied with the OK Button. A tag of the Text Tag 16-Bit Character Set 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 CO_AddressTag.</td>
</tr>
<tr>
<td>2</td>
<td>The table configured in the ex_3_chapter_01c.pdl picture consists of multiple single objects. All properties of these objects - that are to be changed - are equipped with a Tag Connection to one of the color tags.</td>
</tr>
</tbody>
</table>
Step | Procedure: Implementation in the Graphics Designer
---|---
3 | A picture is available, in which the layout of the first dialog window has been created. This is the \textit{ex\_10\_FD\_00.pdl} picture. For each property that can be set, a \textit{Standard Object} $\rightarrow$ \textit{Static Text} and a \textit{Standard Object} $\rightarrow$ \textit{Rectangle} is configured.

The rectangle displays the currently set color of each property. This is realized via a \textit{Tag Connection} to the corresponding color tag. Via a \textit{C-Action at Events} $\rightarrow$ \textit{Mouse} $\rightarrow$ \textit{Press left at the Rectangles and the Static Texts}, the name of the corresponding color tag is written to the address tag.

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 \textit{Properties} $\rightarrow$ \textit{Geometry} $\rightarrow$ \textit{Position X of the Graphic Object1}.

![Image of the Table Color dialog](image)

4 | The layout of the second dialog window has been configured in another picture. This is the \textit{ex\_10\_FD\_01.pdl} picture. For each selectable color, a \textit{Standard Object} \textit{Rectangle} is configured, whose background color corresponds to the color to be set.

At \textit{Events} $\rightarrow$ \textit{Mouse} $\rightarrow$ \textit{Press left}, a \textit{Direct Connection} is created for each rectangle. This connection switches the value of the \textit{Background Color property} of the corresponding object to the color tag contained in the address tag.

![Image of the Direct Connection dialog](image)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>At Events ➔ <em>Mouse ➔ Mouse Action</em>, a <strong>Direct Connection</strong> is created for each <strong>Rectangle</strong> that makes the current window invisible. Via a separate <strong>Windows Object ➔ Button</strong>, the next dialog can be opened.</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="Palette" /></td>
</tr>
<tr>
<td>6</td>
<td>The layout of the third dialog window has been configured in another picture. This is the <code>ex_10_FD_03.pdl</code> picture. Just as described in step 4, a <strong>Standard Object ➔ Rectangle</strong> is configured for each color that can be selected. The <strong>Direct Connection at Events ➔ Mouse ➔ Press left</strong>, however, does not describe the color tag indicated by the address tag, but the temporary color tag <code>CO_TEMP</code>. The value contained in this tag is only written to the color tag to be processed after the OK button has been pressed. In the picture, a <strong>Smart Object ➔ Graphic Object</strong> has been configured, which displays the color value currently contained in the <code>CO_TEMP</code> tag. In the sample, this is the <strong>Selection object</strong>. The position of this object is changed upon the selection of a rectangle via a <strong>C-Action at Events ➔ Mouse ➔ Press left</strong>.</td>
</tr>
<tr>
<td></td>
<td><img src="image2" alt="Colors" /></td>
</tr>
</tbody>
</table>

![Diagram](image3)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>In the <code>ex_3_chapter_01c.pdl</code> picture, a Smart Object → Picture Window has been configured for each dialog. The Picture Window contained in the first dialog is opened via a Windows Object → Button.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Implementation in the Graphics Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Three pictures are available, who only consist of a <strong>Smart Object</strong> ➔ <strong>Graphic Object</strong>. These are the <code>ex_10_BH_00.pdl</code>, <code>ex_10_BH_01.pdl</code> and <code>ex_10_BH_02.pdl</code> pictures. In each of the <strong>Graphic Objects</strong>, bitmaps are displayed representing the bar graph background.</td>
</tr>
<tr>
<td>2</td>
<td>In the <code>ex_3_chapter_01e.pdl</code> picture, three <strong>Smart Objects</strong> ➔ <strong>Status Displays</strong> 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.</td>
</tr>
</tbody>
</table>
3 Above each Status Display, a Smart Object Picture Window is placed, with the pictures described in step 1 set at Properties Miscellaneous Picture Name.

At Properties Geometry Window Height, a Dynamic Dialog each is created, which controls the height of the picture window depending on the value of the tag to be displayed.

Only positive values can be displayed. Therefore, the absolute value of the tag is formed with the abs() function. The maximum value to be displayed is 100. Since the Picture Window 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.

4 Via three Smart Objects Status Displays on the toolbar, the bar graphs can be deactivated. This is performed by a C-Action each at Events Mouse Press left. This C-Action switches the visibility of the Picture Window, switches the picture containing the bar graph status display and switches the picture displayed by the own object.
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
    Integrate, 4-89
  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
Index 09.99

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