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.)
# Table of contents

1 **Configuration Manual**............................................................................. 1-1  
   1.1 Configuration Manual - Notes regarding Structure and Application..... 1-2  

2 **WinCC - General Information**........................................................... 2-1  
   2.1 WinCC - the Concept ..................................................................... 2-2  
      2.1.1 The WinCC Interfaces............................................................. 2-3  
   2.2 WinCC - Terms and their Explanations.......................................... 2-5  

3 **Configuration - General Subjects**...................................................... 3-1  
   3.1 Before the Project Start................................................................. 3-2  
   3.2 Specifications in Detail .................................................................. 3-3  
      3.2.1 Specification: WinCC Project Name....................................... 3-4  
      3.2.2 Specification: Tag Names ..................................................... 3-5  
      3.2.3 Specification: Picture Names ............................................... 3-7  
      3.2.4 Specification: Scripts and Actions ....................................... 3-9  
      3.2.5 Specification: The User Interface ........................................ 3-10  
      3.2.6 Specification: The Control Concept...................................... 3-15  
      3.2.7 Specification: The Color Definition ...................................... 3-17  
      3.2.8 Specification: The Update Cycles ....................................... 3-18  
      3.2.9 Specification: The User Rights ............................................ 3-19  
      3.2.10 Specification: Alarming....................................................... 3-20  
      3.2.11 Specification: For the Implementation.................................... 3-21  
      3.3.1 Update Cycles - How and where are they set.......................... 3-23  
      3.3.1.1 The Update in the Picture................................................... 3-23  
      3.3.1.2 Update Cycle Types ......................................................... 3-25  
      3.3.1.3 Meaning of the Update Cycles ......................................... 3-27  
      3.3.1.4 Information regarding the Application of Update Cycles...... 3-28  
      3.3.1.5 Execution of Background Scripts (Global Script) ................ 3-36  
      3.3.2 Adding Dynamics in WinCC.................................................. 3-40  
      3.3.2.1 Making Properties dynamic.............................................. 3-40  
      3.3.2.2 Making Events dynamic................................................... 3-41  
      3.3.2.3 Dynamization Types for Objects ...................................... 3-41  
      3.3.3 WinCC System Environment.................................................. 3-45  
      3.3.3.1 Folder Structure of the WinCC System............................. 3-45  
      3.3.4 WinCC Project Environment.................................................. 3-48  
      3.3.4.1 WinCC Project - Folder Structure................................... 3-48  
      3.3.5 Automatic Project Start in WinCC ....................................... 3-51
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.6</td>
<td>Coordinated Shut Down of WinCC</td>
<td>3-54</td>
</tr>
<tr>
<td>3.3.6.1</td>
<td>Notes on how to Install a UPS</td>
<td>3-54</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Data Backup</td>
<td>3-55</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Copying a Backed Up WinCC Project to a new Destination Computer</td>
<td>3-57</td>
</tr>
<tr>
<td>3.3.9</td>
<td>Reuse - Transfer of Project Parts to a New or Existing Project</td>
<td>3-59</td>
</tr>
<tr>
<td>3.3.9.1</td>
<td>Transfer of Pictures</td>
<td>3-60</td>
</tr>
<tr>
<td>3.3.9.2</td>
<td>Transfer of Symbols and Bitmaps</td>
<td>3-62</td>
</tr>
<tr>
<td>3.3.9.3</td>
<td>Transferring a Project Library (with preconfigured Symbols and Customized Objects)</td>
<td>3-63</td>
</tr>
<tr>
<td>3.3.9.4</td>
<td>Transfer of Actions</td>
<td>3-65</td>
</tr>
<tr>
<td>3.3.9.5</td>
<td>Transfer of Tags</td>
<td>3-66</td>
</tr>
<tr>
<td>3.3.9.6</td>
<td>Transfer of Multilingual Texts (from Pictures, in Messages)</td>
<td>3-73</td>
</tr>
<tr>
<td>3.3.9.7</td>
<td>Transfer of Messages</td>
<td>3-74</td>
</tr>
<tr>
<td>3.3.9.8</td>
<td>Transfer of Measured Values</td>
<td>3-77</td>
</tr>
<tr>
<td>3.3.9.9</td>
<td>Transfer of Print Layouts</td>
<td>3-77</td>
</tr>
<tr>
<td>3.3.9.10</td>
<td>Transfer of Global Actions</td>
<td>3-77</td>
</tr>
<tr>
<td>3.3.9.11</td>
<td>Transfer of Project Functions</td>
<td>3-77</td>
</tr>
<tr>
<td>3.3.9.12</td>
<td>Application of Standard Functions</td>
<td>3-77</td>
</tr>
<tr>
<td>3.3.9.13</td>
<td>Transfer of the User Administrator</td>
<td>3-77</td>
</tr>
<tr>
<td>3.3.10</td>
<td>Operation without a Mouse</td>
<td>3-78</td>
</tr>
<tr>
<td>3.3.10.1</td>
<td>Operation via Keyboard</td>
<td>3-78</td>
</tr>
<tr>
<td>3.3.10.2</td>
<td>Movement over Control Objects (Input Fields and Control Fields)</td>
<td>3-82</td>
</tr>
<tr>
<td>3.3.10.3</td>
<td>Alarm Logging Function Keys for the Toolbar Buttons</td>
<td>3-84</td>
</tr>
<tr>
<td>3.3.10.4</td>
<td>Alarm Logging - Toolbar Buttons designed specifically for a Plant</td>
<td>3-87</td>
</tr>
<tr>
<td>3.3.10.5</td>
<td>Tag Logging Function Keys for the Toolbar Buttons</td>
<td>3-87</td>
</tr>
<tr>
<td>3.3.10.6</td>
<td>Starting a Print Job</td>
<td>3-90</td>
</tr>
<tr>
<td>3.3.10.7</td>
<td>Logging On or Off</td>
<td>3-91</td>
</tr>
<tr>
<td>3.3.11</td>
<td>Picture Module Technology</td>
<td>3-92</td>
</tr>
<tr>
<td>3.3.11.1</td>
<td>Process Box as Picture Module</td>
<td>3-94</td>
</tr>
<tr>
<td>3.3.11.2</td>
<td>Picture Module with Indirect Addressing</td>
<td>3-96</td>
</tr>
<tr>
<td>3.3.11.3</td>
<td>Customized Objects</td>
<td>3-97</td>
</tr>
<tr>
<td>3.3.11.4</td>
<td>Dynamic Instance</td>
<td>3-98</td>
</tr>
<tr>
<td>3.3.11.5</td>
<td>Prototype Pictures</td>
<td>3-99</td>
</tr>
<tr>
<td>3.3.11.6</td>
<td>OCX Objects</td>
<td>3-103</td>
</tr>
<tr>
<td>3.3.12</td>
<td>Online Configuration (Runtime) - Notes, Limitations</td>
<td>3-104</td>
</tr>
<tr>
<td>4</td>
<td>WinCC C-Course</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1</td>
<td>Development Environment for C-Scripts</td>
<td>4-3</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Action Editor of the Graphics Designer</td>
<td>4-4</td>
</tr>
<tr>
<td>4.1.2</td>
<td>The Global Script Editor</td>
<td>4-11</td>
</tr>
<tr>
<td>4.2</td>
<td>Variables</td>
<td>4-19</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Sample 1 - C Data Types (Integers)</td>
<td>4-21</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.2.2 Sample 2 - Defined Data Types (Integers)</td>
<td>4-23</td>
<td></td>
</tr>
<tr>
<td>4.2.3 Sample 3 - WinCC Tags (Integers)</td>
<td>4-25</td>
<td></td>
</tr>
<tr>
<td>4.2.4 Sample 4 - C Data Types (Floating-Point Numbers)</td>
<td>4-27</td>
<td></td>
</tr>
<tr>
<td>4.2.5 Sample 5 - WinCC Tags (Floating-Point Numbers)</td>
<td>4-28</td>
<td></td>
</tr>
<tr>
<td>4.2.6 Sample 6 - Static and External Variables</td>
<td>4-29</td>
<td></td>
</tr>
<tr>
<td>4.3 Operators and Mathematical Functions in C</td>
<td>4-31</td>
<td></td>
</tr>
<tr>
<td>4.3.1 Sample 1 - Basic Mathematical Operations</td>
<td>4-33</td>
<td></td>
</tr>
<tr>
<td>4.3.2 Sample 2 - Increment and Decrement Operator</td>
<td>4-34</td>
<td></td>
</tr>
<tr>
<td>4.3.3 Sample 3 - Bit Operations</td>
<td>4-36</td>
<td></td>
</tr>
<tr>
<td>4.3.4 Sample 4 - Rotating Byte-by-Byte</td>
<td>4-38</td>
<td></td>
</tr>
<tr>
<td>4.3.5 Sample 5 - Mathematical Functions</td>
<td>4-39</td>
<td></td>
</tr>
<tr>
<td>4.4 Pointers</td>
<td>4-41</td>
<td></td>
</tr>
<tr>
<td>4.4.1 Sample 1 - Pointers</td>
<td>4-43</td>
<td></td>
</tr>
<tr>
<td>4.4.2 Sample 2 - Vectors</td>
<td>4-44</td>
<td></td>
</tr>
<tr>
<td>4.4.3 Sample 3 - Pointers and Vectors</td>
<td>4-45</td>
<td></td>
</tr>
<tr>
<td>4.4.4 Sample 4 - Strings</td>
<td>4-47</td>
<td></td>
</tr>
<tr>
<td>4.4.5 Sample 5 - WinCC Text Tags</td>
<td>4-48</td>
<td></td>
</tr>
<tr>
<td>4.5 Loops and Conditional Statements</td>
<td>4-49</td>
<td></td>
</tr>
<tr>
<td>4.5.1 Sample 1 - while Loop</td>
<td>4-51</td>
<td></td>
</tr>
<tr>
<td>4.5.2 Sample 2 - do-while Loop</td>
<td>4-52</td>
<td></td>
</tr>
<tr>
<td>4.5.3 Sample 3 - for Loop</td>
<td>4-53</td>
<td></td>
</tr>
<tr>
<td>4.5.4 Sample 4 - Endless Loops</td>
<td>4-54</td>
<td></td>
</tr>
<tr>
<td>4.5.5 Sample 5 - if-else Statement</td>
<td>4-56</td>
<td></td>
</tr>
<tr>
<td>4.5.6 Sample 6 - switch-case Statement</td>
<td>4-57</td>
<td></td>
</tr>
<tr>
<td>4.6 Functions</td>
<td>4-58</td>
<td></td>
</tr>
<tr>
<td>4.6.1 Sample 1 - Transfer of Value Parameters</td>
<td>4-59</td>
<td></td>
</tr>
<tr>
<td>4.6.2 Sample 2 - Transfer of Address Parameters</td>
<td>4-61</td>
<td></td>
</tr>
<tr>
<td>4.6.3 Writing to the transferred Address Range</td>
<td>4-63</td>
<td></td>
</tr>
<tr>
<td>4.6.4 Return of the Result Address</td>
<td>4-65</td>
<td></td>
</tr>
<tr>
<td>4.7 Structures</td>
<td>4-68</td>
<td></td>
</tr>
<tr>
<td>4.7.1 Sample 1 - Structure Variable</td>
<td>4-69</td>
<td></td>
</tr>
<tr>
<td>4.7.2 Sample 2 - Type Definition</td>
<td>4-70</td>
<td></td>
</tr>
<tr>
<td>4.7.3 Sample 3 - WinCC Structure Type</td>
<td>4-72</td>
<td></td>
</tr>
<tr>
<td>4.7.4 Sample 4 - Function for Reading a WinCC Structure Type</td>
<td>4-74</td>
<td></td>
</tr>
<tr>
<td>4.8 WinCC API</td>
<td>4-77</td>
<td></td>
</tr>
<tr>
<td>4.8.1 Sample 1 - Changing Properties via an RT Function</td>
<td>4-79</td>
<td></td>
</tr>
<tr>
<td>4.8.2 Sample 2 - Creating a Tag Connection via an RT Function</td>
<td>4-81</td>
<td></td>
</tr>
<tr>
<td>4.8.3 Sample 3 - Creating a New Object via a CS Function</td>
<td>4-83</td>
<td></td>
</tr>
<tr>
<td>4.8.4 Sample 4 - Changing Properties via a CS Function</td>
<td>4-85</td>
<td></td>
</tr>
<tr>
<td>4.8.5 Sample 5 - Creating a Tag Connection via a CS Function</td>
<td>4-87</td>
<td></td>
</tr>
<tr>
<td>4.8.6 Sample 6 - Listing Objects via a CS Function</td>
<td>4-89</td>
<td></td>
</tr>
<tr>
<td>4.9 Project Environment</td>
<td>4-92</td>
<td></td>
</tr>
<tr>
<td>4.9.1 Sample 1 - Determining the Project File</td>
<td>4-93</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.9.2</td>
<td>Sample 2 - Determining the Project Path</td>
<td>4-94</td>
</tr>
<tr>
<td>4.9.3</td>
<td>Sample 3 - Determining the Project Path via a Project Function</td>
<td>4-96</td>
</tr>
<tr>
<td>4.9.4</td>
<td>Sample 4 - Determining the Installation Folder</td>
<td>4-98</td>
</tr>
<tr>
<td>4.9.5</td>
<td>Sample 5 - Determining the Computer Name</td>
<td>4-100</td>
</tr>
<tr>
<td>4.9.6</td>
<td>Sample 6 - Determining the User Name</td>
<td>4-101</td>
</tr>
<tr>
<td>4.10</td>
<td>Windows API</td>
<td>4-102</td>
</tr>
<tr>
<td>4.10.1</td>
<td>Sample 1 - Setting Windows Properties</td>
<td>4-103</td>
</tr>
<tr>
<td>4.10.2</td>
<td>Sample 2 - Reading the System Time</td>
<td>4-104</td>
</tr>
<tr>
<td>4.10.3</td>
<td>Sample 3 - Playing Sound Files</td>
<td>4-105</td>
</tr>
<tr>
<td>4.10.4</td>
<td>Sample 4 - Starting a Program</td>
<td>4-107</td>
</tr>
<tr>
<td>4.11</td>
<td>Standard Dialogs</td>
<td>4-108</td>
</tr>
<tr>
<td>4.11.1</td>
<td>Sample 1 - Language Switch</td>
<td>4-109</td>
</tr>
<tr>
<td>4.11.2</td>
<td>Sample 2 - Tag Selection</td>
<td>4-111</td>
</tr>
<tr>
<td>4.11.3</td>
<td>Sample 3 - Error Box</td>
<td>4-113</td>
</tr>
<tr>
<td>4.11.4</td>
<td>Sample 4 - Question Box</td>
<td>4-114</td>
</tr>
<tr>
<td>4.11.5</td>
<td>Sample 5 - Open Standard Dialog</td>
<td>4-116</td>
</tr>
<tr>
<td>4.12</td>
<td>Files</td>
<td>4-118</td>
</tr>
<tr>
<td>4.12.1</td>
<td>Sample 1 - Securing Data</td>
<td>4-120</td>
</tr>
<tr>
<td>4.12.2</td>
<td>Sample 2 - Reading Data</td>
<td>4-121</td>
</tr>
<tr>
<td>4.12.3</td>
<td>Sample 3 - Reporting</td>
<td>4-122</td>
</tr>
<tr>
<td>4.13</td>
<td>Dynamic Wizard</td>
<td>4-124</td>
</tr>
<tr>
<td>4.13.1</td>
<td>Creation of Dynamic Wizard Functions</td>
<td>4-125</td>
</tr>
<tr>
<td>4.13.2</td>
<td>Structure of a Dynamic Wizard Function</td>
<td>4-126</td>
</tr>
<tr>
<td>5</td>
<td>Appendix</td>
<td>5-1</td>
</tr>
<tr>
<td>5.1</td>
<td>Tips and Tricks</td>
<td>5-2</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Formatted Input/Output at the I/O Field</td>
<td>5-3</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Object-Specific Actions at Open Picture</td>
<td>5-4</td>
</tr>
<tr>
<td>5.1.3</td>
<td>WinCC Scope</td>
<td>5-5</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Access to the Database</td>
<td>5-6</td>
</tr>
<tr>
<td>5.1.4.1</td>
<td>Access to the Database from MS Excel/MS Query</td>
<td>5-6</td>
</tr>
<tr>
<td>5.1.4.2</td>
<td>Access to the Database from MS Access</td>
<td>5-9</td>
</tr>
<tr>
<td>5.1.4.3</td>
<td>Access to the Database from ISQL</td>
<td>5-6</td>
</tr>
<tr>
<td>5.1.4.4</td>
<td>Access to the Database from WinCC Scope</td>
<td>5-10</td>
</tr>
<tr>
<td>5.1.4.5</td>
<td>Export from the Database via C-Actions</td>
<td>5-12</td>
</tr>
<tr>
<td>5.1.4.6</td>
<td>Database Selections</td>
<td>5-13</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Serial Connection</td>
<td>5-14</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Color Table</td>
<td>5-15</td>
</tr>
<tr>
<td>5.2</td>
<td>Documentation of the S5 Alarm System</td>
<td>5-16</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Listing of the Software Blocks</td>
<td>5-17</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Hardware Requirements</td>
<td>5-18</td>
</tr>
</tbody>
</table>
5.2.3 Integration of the S5 Alarm System into the SIMATIC S5 Application Program .................................................. 5-19
  5.2.3.1 Structure of the Offset Data Block ................................................................. 5-22
  5.2.3.2 Basic Message Number .............................................................................. 5-23
  5.2.3.3 Offset Message Number/Signal States of the Messages ......................... 5-24
  5.2.3.4 Signal Status Block ................................................................................... 5-25
  5.2.3.5 Address of the last Signal Status Block .................................................... 5-26
  5.2.3.6 Signal States .................................................................................................. 5-27
  5.2.3.7 Idle States ...................................................................................................... 5-27
  5.2.3.8 Acknowledgment Bits .................................................................................. 5-28
  5.2.3.9 Edge Trigger Flags ..................................................................................... 5-28
  5.2.3.10 Structure of the Parameter Data Block ....................................................... 5-28
  5.2.3.11 Structure of a Message Block ..................................................................... 5-30
  5.2.3.12 Message Number .......................................................................................... 5-30
  5.2.3.13 Message Status ............................................................................................ 5-31
  5.2.3.14 Date/Time Stamp ......................................................................................... 5-31
  5.2.3.15 Process Tag .................................................................................................. 5-31
  5.2.3.16 Job Number/Batch Identifier ..................................................................... 5-31
  5.2.3.17 Reserve .......................................................................................................... 5-31
  5.2.3.18 Generation of a Message Block ................................................................ 5-31
  5.2.3.19 The Internal FIFO Buffer (Ring) .............................................................. 5-32
  5.2.3.20 The Send Mailbox - Data Transfer to the Higher-Level WinCC System .......... 5-32
  5.2.4 Interface Description ....................................................................................... 5-34
    5.2.4.1 System Data Block 80 ............................................................................... 5-34
    5.2.4.2 Offset Data Block ....................................................................................... 5-34
    5.2.4.3 Parameter Data Block ............................................................................... 5-34
    5.2.4.4 Send Mailbox/Transfer Mailbox ................................................................. 5-34
  5.2.5 Assigning Parameters to the S5 Alarm System/System DB 80 ...................... 5-35
  5.2.6 Configuration Sample for the S5 Alarm System ........................................... 5-40
    5.2.6.1 DB 80 Parameterization ............................................................................ 5-40
    5.2.6.2 Setup of the Data Blocks ............................................................................ 5-41
    5.2.6.3 Initialization of the Offset Data Blocks ..................................................... 5-41
  5.2.7 Documentation of the SIMATIC S5 Command Blocks .................................. 5-45
    5.2.7.1 Listing of the Software Blocks .................................................................. 5-45
    5.2.7.2 Hardware Requirements ........................................................................... 5-46
    5.2.7.3 Call Parameters of the FB 87: EXECUTE .................................................. 5-46
  5.2.8 Interface Description ....................................................................................... 5-47
    5.2.8.1 Configuration Sample for the S5 Command Blocks .................................... 5-49
  5.2.9 Task and Function of the S5 Time Synchronization ........................................ 5-50
    5.2.9.1 Listing of the Software Blocks .................................................................. 5-50
    5.2.9.2 Hardware Requirements ........................................................................... 5-50
    5.2.10 Call Parameters of the FB 86: MESS:CLOCK ............................................. 5-51
5.2.11 Data Formats for Date and Time .................................................... 5-53
5.2.11.1 Clock Data Area CPU 944, CPU 945 ........................................ 5-54
5.2.11.2 Clock Data Area CPU 928B, CPU 948 ................................. 5-55
5.2.11.3 Clock Data Area CPU 946, CPU 947 ........................................ 5-56
5.2.11.4 Clock Data Format for Message Blocks ................................. 5-57
5.2.12 Interface Description ................................................................. 5-58
5.2.13 Interaction with the WinCC Alarm System ................................. 5-59

5.3 Format DLL Interface to Alarm Logging and Tag Logging .......... 5-60
5.3.1 Shared Interface to Alarm Logging and Tag Logging ................ 5-61
5.3.2 Tag Logging-specific Additions .................................................... 5-63
5.3.3 API Functions of a WinCC Format DLL ....................................... 5-64
5.3.3.1 Initialization of the Format DLL ............................................... 5-64
5.3.3.2 Query of the Properties of a Format DLL ............................... 5-65
5.3.3.3 Query of the Name of the Format DLL .................................... 5-67

5.3.4 Shut Down of the Format DLL ..................................................... 5-68
5.3.4.1 Expansions to the Configuration .............................................. 5-68
5.3.4.2 Dialog Expansion during the Configuration of S7PMC Messages. 5-68
5.3.4.3 Dialog Expansion during the Configuration of Archive Tags .... 5-71
5.3.4.4 Online Services ........................................................................ 5-72
5.3.4.5 Registration of all Archive Tags .............................................. 5-73
5.3.4.6 Language Switch ...................................................................... 5-75

5.3.5 Formatting .................................................................................. 5-76
5.3.5.1 Derivation of Single Messages ............................................... 5-76
5.3.5.2 Acknowledging, Locking/Enabling Messages ............................ 5-77
5.3.5.3 Processing in the Event of a Status Change ............................. 5-78
5.3.5.4 Message Update of the S7PMC Format DLL ......................... 5-78
5.3.5.5 Formatting of Archive Tags ..................................................... 5-79
5.3.5.6 Derivation of individual Archive Tag Values ......................... 5-79
5.3.5.7 Locking/Enabling of the Archive Tags ..................................... 5-80
5.3.5.8 Processing in the Event of a Status Change ............................. 5-80

5.4 Global Library .............................................................................. 5-81
5.4.1 System Blocks ............................................................................ 5-82
5.4.1.1 Motors .................................................................................... 5-82
5.4.1.2 PC/PLC .................................................................................. 5-83
5.4.1.3 Pumps .................................................................................... 5-83
5.4.1.4 Pipes ..................................................................................... 5-84
5.4.1.5 Pipes - Customized Objects ................................................. 5-84
5.4.1.6 Tanks .................................................................................... 5-85
5.4.1.7 Valves - Customized Objects ................................................. 5-85
5.4.1.8 Valves .................................................................................... 5-85

5.4.2 Displays ....................................................................................... 5-86
5.4.2.1 Displays .................................................................................. 5-86
5.4.2.2 Windows ................................................................................. 5-86
5.4.2.3 Scaling ................................................................................... 5-86
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.2.4</td>
<td>Text Fields</td>
<td>5-86</td>
</tr>
<tr>
<td>5.4.2.5</td>
<td>Meters</td>
<td>5-87</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Controls</td>
<td>5-88</td>
</tr>
<tr>
<td>5.4.3.1</td>
<td>3D Buttons</td>
<td>5-88</td>
</tr>
<tr>
<td>5.4.3.2</td>
<td>Control Panels</td>
<td>5-88</td>
</tr>
<tr>
<td>5.4.3.3</td>
<td>Picture Buttons</td>
<td>5-89</td>
</tr>
<tr>
<td>5.4.3.4</td>
<td>Picture Navigation</td>
<td>5-89</td>
</tr>
<tr>
<td>5.4.3.5</td>
<td>Increment/Decrement Buttons</td>
<td>5-89</td>
</tr>
<tr>
<td>5.4.3.6</td>
<td>Controllers</td>
<td>5-90</td>
</tr>
<tr>
<td>5.4.3.7</td>
<td>Language Switch</td>
<td>5-90</td>
</tr>
<tr>
<td>5.4.3.8</td>
<td>Keyboards</td>
<td>5-90</td>
</tr>
<tr>
<td>5.4.3.9</td>
<td>Shift Buttons</td>
<td>5-91</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Symbols</td>
<td>5-92</td>
</tr>
<tr>
<td>5.4.4.1</td>
<td>Shut-Off Devices</td>
<td>5-92</td>
</tr>
<tr>
<td>5.4.4.2</td>
<td>Shut-Off Valves</td>
<td>5-93</td>
</tr>
<tr>
<td>5.4.4.3</td>
<td>DIN 30600</td>
<td>5-94</td>
</tr>
<tr>
<td>5.4.4.4</td>
<td>E Symbols</td>
<td>5-95</td>
</tr>
<tr>
<td>5.4.4.5</td>
<td>Conveyors</td>
<td>5-96</td>
</tr>
<tr>
<td>5.4.4.6</td>
<td>ISA Symbols</td>
<td>5-97</td>
</tr>
<tr>
<td>5.4.4.7</td>
<td>Motors</td>
<td>5-101</td>
</tr>
<tr>
<td>5.4.4.8</td>
<td>Valves</td>
<td>5-102</td>
</tr>
<tr>
<td>5.4.4.9</td>
<td>Miscellaneous 1</td>
<td>5-103</td>
</tr>
<tr>
<td>5.4.4.10</td>
<td>Miscellaneous 2</td>
<td>5-104</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:

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 Configuration Manual

The Configuration Manual is part of the WinCC documentation and is mainly concerned with the practical application of WinCC in projects.

Introduction

In recent years, the demands made on systems for monitoring and controlling production processes as well as for archiving and further processing of production data have risen greatly. In order to meet these new demands, new HMI systems have been developed over the past years.

One of these new systems is WinCC. With respect to functionality, openness and being state of the art, WinCC is without a doubt unique. Older generation HMI systems often provided only one route to solving a specified task. With WinCC, you almost always have a number of different options available to implement tasks. This Configuration Manual has been written to ensure that you always apply the best solution with respect to performance and the amount of configuration work required. This description is intended to provide you with suggested solutions for achieving the most effective use of WinCC in plant projects.

We have implemented these suggested solutions in WinCC sample projects. These sample projects are supplied together with the WinCC CD-ROM. You can use these suggested solutions directly in your own projects and save valuable time in the process.
1.1 Configuration Manual - Notes regarding Structure and Application

Requirements

Before starting work with this Configuration Manual, you should already have some practical experience using WinCC. Newcomers to WinCC will find the Getting Started manual an ideal starting point to learn and become familiar with WinCC. The Getting Started explains the main subjects and functions by means of a small demonstration sample. This Configuration Manual is a supplement to the WinCC Help system (online and documentation). If not explained in this Configuration Manual, you can look up the special features of objects, properties and other subjects in the Help system.

Content and Structure

The manual is subdivided into the following sections:

- **WinCC - the Concept**
  This section contains general information about the WinCC system.

- **Configuration - General Subjects**
  This section contains general and specific information about planning and effectively managing HMI projects.

- **Starting Up the Samples**
  This section contains information about starting up the samples that are created in this manual.

- **WinCC C-Course**
  This section contains a WinCC C-Course. For newcomers, the main rules for using the WinCC script language are described. The C-expert will find a description of the special features of the WinCC development environment.

- **Tag/Variable Configuration**
  This section describes the Project_TagHandling sample. In this sample project, the general handling of tags and simple input/output elements is described.

- **Picture Configuration**
  This section describes the Project_CreatePicture sample. In this sample project, the general handling of WinCC pictures is described.

- **WinCC Editors**
  This section describes the Project_WinCCEditors sample. In this sample project, the editors Tag Logging, Alarm Logging and Report Designer are described.

- **User Archives**
  This section describes the Project_UserArchive sample. In this sample project, the User Archives editor is described.

- **New Function Description**
  This section describes the option of configuring distributed systems that has been added in WinCC V5.

- **Multi-Client**
  This section describes the application of the multi-client project type by means of sample projects.
• Distributed Servers
  This section describes the creation of a WinCC project distributed across multiple servers by means of sample projects.

• Redundancy
  This section describes the configuration of a redundant server pair by means of a sample project.

• Appendix
  This section deals with various additional topics. These topics originate from, among other things, WinCC Solutions and WinCC Tips Tricks.

Conventions

The Configuration Manual uses the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![LMB]</td>
<td>Denotes an operation using the left mouse button.</td>
</tr>
<tr>
<td>![RMB]</td>
<td>Denotes an operation using the right mouse button.</td>
</tr>
<tr>
<td>![DMLB]</td>
<td>Denotes an operation using a double-click of the left mouse button.</td>
</tr>
<tr>
<td>Italic</td>
<td>Denotes terms of the WinCC environment and terms referring to the elements of the program’s interface.</td>
</tr>
<tr>
<td>Italic, Green</td>
<td>Denotes an operating sequence or entry to be followed by the user (color visible only in the online document).</td>
</tr>
<tr>
<td>Blue</td>
<td>Cross references (links) are in blue (color visible only in the online document).</td>
</tr>
</tbody>
</table>

Finding Information

In the printed version of the configuration manual, information can be found in the following ways:

• The **table of contents** lists information organized by topic.
• The **index** lists information organized by key word.

In the online document, information can be found in the following ways:

• The Contents tab lists information organized by topic.
• The Index tab lists information organized by key word.
• The Find tab allows you to search for words in the entire document.

The sample projects described in this manual can directly be copied from the online document to your hard disk drive.
2 WinCC - General Information
2.1 WinCC - the Concept

- In general, there are three solution approaches in WinCC from the configuration viewpoint:
- The configuration using standard WinCC resources
- The usage of existing Windows applications with WinCC via DDE, OLE, ODBC and ActiveX
- Development of your own applications (in VisualC++ or Visual Basic) embedded into WinCC.

For some, WinCC is the HMI system for inexpensive and quick configurations, while for others it is an infinitely expandable system platform. Thanks to the modularity and flexibility of WinCC, totally new possibilities are opened up for planning and implementing automation tasks.

The Operating System: The Basis of WinCC

WinCC is based on the 32-Bit operating system from Microsoft (Windows NT 4.0). This operating system is the standard operating systems on the PC platform.

The modular Structure of WinCC

WinCC offers system modules for visualizing, reporting, acquiring and archiving process data as well as for the coordinated integration of freely formulated user routines. In addition, you can also incorporate your own modules.
2.1.1 The WinCC Interfaces

The Openness of WinCC

WinCC is absolutely open to any form of extension added by the user. This absolute openness is achieved through the modular structure of WinCC and its powerful programming interface. The following figure illustrates the possibilities for interfacing different applications.

Integration of Applications into WinCC

Of crucial importance is the fact that WinCC offers methods for integrating other applications and application blocks homogeneously into the user interface used for process control. As illustrated below, OLE application windows as well as OLE Custom Controls (32-Bit OCX objects) or ActiveX Controls can be integrated into the WinCC applications as if they were true WinCC objects.
Data Management in WinCC

In the following chart, WinCC comprises the entire central section. The graphic shows that the default database Sybase SQL Anywhere is subordinate to WinCC. It is used to file (transaction-protected) all list-oriented configuration data such as tag lists and message texts, but also current process data such as messages, measured values and user data records. This database functions as a server. WinCC can access the database via ODBC, but also via the open programming interface (C-API) as a client.

The same rights are, of course, also granted to other programs. For this reason, a Windows spreadsheet or a Windows database has direct access to the data resources of the WinCC database, irrespective of whether the application is run on the same computer or on a networked workstation. With the aid of the database query language SQL and corresponding connectivity tools (e.g. ODBC drivers), other clients (e.g. UNIX databases such as Oracle, Informix, Ingres) also enjoy access to WinCC's data resources. And, of course, vice versa too. All in all, there is nothing that stands in the way of WinCC being integrated into a factory-wide or corporate concept.
2.2 WinCC - Terms and their Explanations

This section contains an alphabetically ordered list of terms related to WinCC. You will probably already be familiar with many of the terms explained here.

HMI Human Machine Interface
PLC Programmable Logic Controller
CS Configuration System
RT Runtime
3 Configuration - General Subjects

In this section, you will find a great deal of information, instructions and ideas on how to manage projects using WinCC. Some of this information is not specific to WinCC. In the ideal case, these configuration rules should be a style guide for the configuration and design of the runtime projects.
3.1 Before the Project Start

Before you begin with the configuration, you should lay down a number of specifications and conduct some structuring work. This
- simplifies the configuration
- improves the clarity of the project
- simplifies working as a team
- improves stability and performance
- simplifies project maintenance

Clear specifications of the structural guidelines are basic prerequisites for setting up or expanding a corporate standard.

These specifications can be divided into two categories:

Specifications for the Configuration

- Before the project start, the following specifications should be defined:
  - the name of the WinCC project
  - the names of the tags
  - the names of the WinCC screens
  - the rules for creating scripts and actions
  - the configuration rules (corporate standards, library function, working in a team)
  - the mode and method of documenting the project

Specifications for the Runtime Project

Specifications that concern the runtime project (result of configuration). These specifications depend heavily on the application area (e.g. automobile industry, chemical industry, machine manufacturers, etc.). The following specifications should be defined:
- the user interface (screen arrangement, font and font size, runtime language, display of objects, etc.)
- the control concept (screen hierarchy, control philosophy, user rights, valid keys, etc.)
- the colors used for messages, limit values, stati, text, etc.
- the communication modes (type of connection, type and cycles for updating, etc.)
- the quantity schedule (number of alarms, archive values, trends, clients, etc.)
- the message and archiving methods
3.2 Specifications in Detail

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

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

Default Values of the Configuration Tools

In most of WinCC’s editors, certain properties can be set as default values. In this way, WinCC supports your own particular style of configuration and can therefore be optimally configured for specific tasks.

Note:
An example of this are the options that can be set at Graphics Designer - Tools - Settings. A detailed description of this topic can be found in the Online Help of the Graphics Designer.
3.2.1 Specification: WinCC Project Name

General Information

The project name is also suggested as the default name for the folder in which all the data belonging to the WinCC project is stored. You can change the folder name during the initial creation of the project or at a later time (form the Windows Explorer).

Parameters / Limits

All characters, except for some special characters (e.g. \ ? ' ; : / ), are permitted. Numerical values from 0 - 9 are also permissible.

Specification

For the sample projects, described in the second part of the configuration manual, the following applies to the project name:

a...a nn

where:

a    type designation (a-z, A-Z, no special characters)
_n   serial number to distinguish between a number of projects of the same type (numbers 0 - 9), range 00 - 99

Example: cours_00.mcp, or pictu_01.mcp

Note for the General Application

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

Note:

When updating documentation, the WinCC project name can be included in the printouts. This makes it easier to associate and find information.
3.2.2 Specification: Tag Names

General Information

Tag names are no longer restricted to a maximum of 8 characters. Despite this, you should avoid making them too long. If you adhere to strict rules when allocating tag names, you will find this to be tremendously advantageous during configuration.

When creating WinCC projects, the structure of Tag Management is one of the key tasks necessary to ensure quick and effective configuration and high-performance processing during runtime (in scripts).

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

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

Specification

The following applies to the tag names of the sample projects dealt with in this manual:

```
xxx_y...z_a...a_nn
```

where:

- **x** Abbr.
- **Type**

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN</td>
<td>Binary Tag</td>
</tr>
<tr>
<td>U08</td>
<td>Unsigned 8-Bit Value (unsigned)</td>
</tr>
<tr>
<td>S08</td>
<td>Signed 8-Bit Value (signed)</td>
</tr>
<tr>
<td>U16</td>
<td>Unsigned 16-Bit Value</td>
</tr>
<tr>
<td>S16</td>
<td>Signed 16-Bit Value</td>
</tr>
<tr>
<td>U32</td>
<td>Unsigned 32-Bit Value</td>
</tr>
<tr>
<td>S32</td>
<td>Signed 32-Bit Value</td>
</tr>
<tr>
<td>G32</td>
<td>Floating-Point Number 32-Bit IEEE 754</td>
</tr>
<tr>
<td>G64</td>
<td>Floating-Point Number 64-Bit IEEE 754</td>
</tr>
<tr>
<td>T08</td>
<td>Text Tag 8-Bit Character Set</td>
</tr>
<tr>
<td>T16</td>
<td>Text Tag 16-Bit Character Set</td>
</tr>
<tr>
<td>RAW</td>
<td>Raw Data Type</td>
</tr>
<tr>
<td>TER</td>
<td>Text Reference</td>
</tr>
<tr>
<td>STU</td>
<td>Structure Types</td>
</tr>
</tbody>
</table>
Y  Abbr.  Origin
r   Pure read tag from the PLC (read)
w   Write and read tag from the PLC (write)
i   Internal WinCC tag without link to the PLC
x   Tag with indirect addressing (a text tag whose content is a tag name)
_z  Group (corresponds to a plant section or building)
Paint ... e.g. name of a plant section
_a  Tag Name (e.g. Measurement Point Name)
_EU0815V10 ... e.g. name of a measurement point
_n  Serial number of the instance (numbers 0 - 9), range 00 - 99

Parameters / Limits

• For the assignment of tag names, the following limitations should be observed:
• The special character @ should be reserved for WinCC system tags (however, the use of this character elsewhere is not prohibited).
• The special characters ’ and % must not be used.
• The special character " and the character string // should not be used, since they have a special meaning in C-Scripts (introduction/conclusion of a character string as well as the introduction of a comment).
• No spaces.
• No distinction is made between upper case and lower case letters in tag names.

Note for the General Application

The tag names assigned in our examples are only suggestions.
When using tags in scripts and Excel, you may find it useful to adhere to a fixed length for the separate parts of a tag name (if necessary, using 0 or x as a fill character).
Large quantities of tags can be created and maintained very effectively and simply in, for example, Excel. If the tag names have a fixed structure, it is considerably easier to create the tag lists in Excel. These tag lists created in Excel can then be imported into the current WinCC project using the program \SmartTools\CC_TagImportExport\Var_exim.exe, which is located on the WinCC CD-ROM.
3.2.3 Specification: Picture Names

General Information

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

Parameters / Limits

- For the assignment of picture names, the following limitations should be observed:
- Maximum length of 255 characters
- Any character with the exception of certain special characters
- No distinction is made between upper case and lower case letters in picture names

Specification

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

aaaaa_k_x...x_nn

where:

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

_k Identification of the Picture Type

_0 Start Picture
_1 Overview Picture
_2 Button Picture
_3 Plant Picture
_4 Detail Picture
_5 Message Picture
_6 Trend Picture
_7 ...
_8 ...
_9 Diagnostics Pictures (for testing or commissioning only)

_x Name for the description of the picture function (a-z, A-Z, no special characters), maximum of 30 characters.
chapter ... e.g. name of the chapters of the C-Course

_n Serial number of the type (number 0 - 9), range 0 - 99
Note for the General Application

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

General Information

You can create your own scripts and actions in WinCC projects. The names you assign should be of an expressive nature. This makes things a lot easier when using scripts later. Using a proportional font tends to be a nuisance when configuring in the Global Script editor. For this reason, choose a font with a constant character width (e.g. Courier) to make things more readable.

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

Specification

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

We use the proportional font *Courier New*, font size 8
All variable names and comments are in English

Note for the General Application

A detailed description on how to use the scripts, actions and editors can be found in the chapter Development Environment for C-Scripts.
3.2.5 Specification: The User Interface

General Information

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

These are the screens viewed daily by the operator (customer). In a screen display system, information about the current status of the plant is presented to the users solely by means of the pictures displayed. This interface must therefore provide information in a manner as comprehensive and understandable as possible.

WinCC allows you to configure the user interface precisely as needed. How you lay out the user interface of your own particular system depends on the hardware used, on the demands during processing and on already existing specifications.

The Users

When configuring the user interface, the attention should be focused on the users, for whom the configuration is after all being performed.

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

The users need as much information as possible. Using this data as the foundation, the users make the decisions that are essential to keeping the process running and at high level of quality. The main job of the users is not to respond to alarms (the process has then already been thrown off balance), but to use their experience and knowledge of the process and the information provided by the operating system to predict the direction in which the process is developing. The users should be able to counteract irregularities before they arise.

WinCC gives you the ability to edit and display this information to the users effectively.

How much Information should be placed into a Picture?

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

- If a picture contains too much information, it will be difficult to read and the search for information will take longer. The probability of errors being made by the users is also increased.

If a picture contains too little information, the amount of work the users have to do is increased. They lose track of the process and have to change pictures frequently in order to find the information required. This leads to delayed responses, control inputs and instability of the process being controlled.
Studies have shown that experienced users want **as much information as possible in every picture**, so that they do not have to change pictures as frequently. In contrast, beginners become confused and uncertain of what to do when a lot of information is packed into one picture. They either cannot find the right information or cannot find it in time. But experience has taught us one thing: A beginner soon becomes experienced, but an experienced user will never become inexperienced again.

**Hiding Information**

The information displayed should be important and easily understood. Certain information can remain hidden (e.g. measurement point identifiers) until they are needed.

**Displaying Information**

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

**Color Coding**

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

**Displaying Text**

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

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

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

Screen Layout

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

Picture occupying the entire Screen Area

Screen sectioned into the Overview, Button and Plant Pictures
Example of an Operator Panel

Parameters / Limits

The size of the individual pictures can be set as needed within fixed boundaries (min. 1 x 1, max. 4096 x 4096 pixels). In the case of a single-user system with a 17" monitor, we recommend a maximum resolution of 1024 x 768 pixels. With multi-user systems (multi-VGA), you may find a higher resolution useful.

In the case of operator panels, the technology used usually restricts the resolution available (TFT from 640 x 480 through 1024 x 768 pixels).

Specification

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

Resolution

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

Texts

The names of measurement points are written in Courier, pure descriptions, all other texts and text displays in Arial. For message boxes following the Windows style, the fonts MS Sans Serif and System are used.

The font size is adjusted as necessary.
Information in the Picture

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

Screen Layout

We will configure the basic options for laying out the screen. In the remaining projects, however, we will apply the method of sectioning the screen into a header, workspace and footer.

Note for the General Application

You can reuse the basic layout of the concepts applied for your own projects.
3.2.6 Specification: The Control Concept

General Information

You control your process application under WinCC using the usual input devices such as a keyboard, mouse, touch screen or industrial joystick. If your computer is located in an industrial setting with extreme conditions where it would be impossible to use a mouse, you can configure tab orders and the alpha cursor. The tab orders move you through controllable fields, while the alpha cursor moves you to the input fields. Each control action can be locked against unauthorized access.

Opening Pictures

The concept for the selection of the screens depends on several factors. Of utmost importance is the number of pictures and the structure of the process which is to be displayed.

In small applications, the pictures can be arranged as a ring or FIFO buffer.

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

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

Hierarchy

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

Layer 1

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

Layer 2

Categorized under layer 2 are the process pictures.
This layer contains detailed information about a specific process section and shows which plant objects belong to this process section.
This layer also indicates which plant object the alarm refers to.
Layer 3

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

Specification

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

Note for the General Application

Our projects should only be regarded as suggestions for creating your own control concept. When extending plants, you must take the existing control concepts into account. Many users will find that their company already has corporate conventions and standards that must be adhered to when making configurations.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The optional WinCC package Basic Process Control offers a ready-made control concept.</td>
</tr>
<tr>
<td>This optional package also contains other useful and powerful functions (e.g. storage).</td>
</tr>
</tbody>
</table>
3.2.7 Specification: The Color Definition

General Information

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

• Colors should always be defined for the following areas. The colors can be defined in accordance with DIN EN 60073, which corresponds to VDE 0199, but must always be agreed on together with the user:
  • Colors for messages (activated / cleared / acknowledged)
  • Colors for stati (on / off / faulty)
  • Colors for character objects (circuits / fill levels)
  • Colors for warning and limit values

Specification

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

The color setting of your PC must be set to greater than 256 colors for the sample projects to be displayed correctly.

To make it easier for you to get oriented, we will use a different background color for the separate topics (tags, C-Course, picture configuration) dealt with in the sample projects. The background color in the overview and button sections is darker.

For the alarm system, each message class and message type assigned to a message class is given a certain color code.

Note for the General Application

After defining the colors, you should, if necessary, adjust the default settings of WinCC. A table for coding the color values in C-Actions can be found in the appendix, chapter Color Table.
3.2.8 Specification: The Update Cycles

General Information

When determining the update cycles, always consider the system as a whole: What is updated and how often updating is carried out. Choosing the incorrect update cycles can have negative effects on the performance of the HMI system. When looking at an overall system (PLC - communication - HMI), changes should be detected where they occur, namely in the process (PLC). In many cases, it is the bus system that poses the bottleneck for data transmission. When specifying the mode of updating measured values, you must pay attention to how quickly the measured value actually changes. For the temperature control of a boiler with a volume of 5,000 liters, the update of the actual value in 500 ms intervals makes no sense.

32-Bit HMI System

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

Specification

The following specifications apply to updating in the projects dealt with in this manual: Insofar as the task definition permits, updating is performed driven by events. Since we predominantly work with internal tags, we often trigger upon the change of the tag. When using external tags, this can lead to increased system loads depending on the process driver connection. If the communication permits an event-driven transfer, it should be selected for time-critical data. Uncritical data can then be retrieved by the HMI in appropriate cycles (polling procedure).

Note for the General Application

A detailed description about the application of update cycles can be found in the chapter Update Cycles - How and where are they set.
3.2.9 Specification: The User Rights

General Information

When operating plant, it is necessary to protect certain operator functions against unauthorized access. A further requirement is that only certain persons have access to the configuration system.

You can specify users and user groups and define various authorization levels in the User Administrator. These authorization levels can be linked to the control elements in the pictures.

The user groups and users can be assigned different authorization levels on an individual basis.

Specification

In the sample projects Project_C-Course and Project_TagHandling, each user is authorized to control the operation of the project.

In the sample project Project_CreatePicture, users can only control the operation of the project after logging in. The password is the same as the project name (Project_CreatePicture).

The buttons used for selecting the individual topics are linked to the authorization level known as Project Control.

Note for the General Application

A description on how to assign user rights can be found in the second part of the configuration manual in the sample project Project_CreatePicture, chapter User Authorizations.
3.2.10 Specification: Alarming

General Information

In general, WinCC supports two alarming procedures:

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

What is to be reported?

When specifying which events and stati are to be reported, many people follow what they see as the safest route and set the software to report all events and changes in status. This leaves it up to the users to decide which messages they look at first.

If too many events are reported in a plant, experience shows us that important messages are only picked once it is too late.

Note for the General Application

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

General Information

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

Note:

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

Specifying Folders

In addition to the folders that are created by WinCC, create additional folders for Word, Excel and temporary files if required.
3.3 Peculiarities during the Configuration with WinCC

The following chapters deal with topics which cut across all aspects of the configuration using WinCC.
These topics supplement the online help of WinCC.
3.3.1 Update Cycles - How and where are they set

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

- the picture structure
- the update of the objects in the currently open picture at the visualization station (Graphics Designer)
- the processing of background scripts (*Global Script*)
- the activation of the data manager and the process communication

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

Data Manager

The current tag values are requested by the data manager - the main administrator of tag management - in accordance with the update cycles set. See chapter Adding Dynamics in WinCC.

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

3.3.1.1 The Update in the Picture

Picture Update

Updating the individual object properties in the picture refers to the objects that have been made dynamic after the picture is opened. The task of the updating cycle is to establish the current status of the particular object in the picture. The update cycle of objects that have been made dynamic can be set by the configurator or by the system for the following types of dynamics:

<table>
<thead>
<tr>
<th>Type of Dynamic</th>
<th>Default Setting</th>
<th>Configuration Customization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Dialog</td>
<td>Tag trigger 2 sec. or Event trigger (e.g. control)</td>
<td>Customization of the time cycles</td>
</tr>
</tbody>
</table>
| Dynamic Wizard      | • You can choose from the following depending on the type of dynamic:  
|                     |   • Event trigger  
|                     |   • Time cycle  
<p>|                     |   • Tag trigger | Customization of the time cycles, events or tags |
| Direct Connection   | Event trigger                                      |                             |
| Tag Connection      | Tag trigger 2 sec.                                 | Customization of the time cycles |</p>
<table>
<thead>
<tr>
<th>Type of Dynamic</th>
<th>Default Setting</th>
<th>Configuration Customization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Dialog</td>
<td>Tag trigger 2 sec.</td>
<td>Customization of the time cycles, tag triggers</td>
</tr>
</tbody>
</table>
| C-Action for Properties | Time cycle 2 sec.     | Customization of the time cycles, tag triggers  
|                         |                       | Direct reading from the PLC                   |
| Object Property         | Setting depends on the dynamic | Editing the update cycle column               |

The update cycles to be selected are specified by WinCC and can be supplemented by user-defined time cycles.

Selecting the update cycles, e.g. for the property of an object:
3.3.1.2 Update Cycle Types

For the update cycles, the following differentiations are made:

<table>
<thead>
<tr>
<th>Type</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Cycle</td>
<td>Time cycle of 2 seconds</td>
</tr>
<tr>
<td>Time Cycle</td>
<td>2 seconds</td>
</tr>
<tr>
<td>Tag Trigger</td>
<td>2 seconds</td>
</tr>
<tr>
<td>Picture Cycle</td>
<td>2 seconds</td>
</tr>
<tr>
<td>Window Cycle</td>
<td>Upon Change</td>
</tr>
<tr>
<td>User-Defined Time Cycles</td>
<td>User cycle 1: 2 sec.</td>
</tr>
<tr>
<td></td>
<td>User cycle 2: 3 sec.</td>
</tr>
<tr>
<td></td>
<td>User cycle 3: 4 sec.</td>
</tr>
<tr>
<td></td>
<td>User cycle 4: 5 sec.</td>
</tr>
<tr>
<td></td>
<td>User cycle 5: 10 sec.</td>
</tr>
</tbody>
</table>
**User Cycle**

Up to 5 *project-related* user cycles can be defined. If the project name is selected in the left tree structure of the WinCC Explorer, clicking on the toolbar button displayed below will open the *Project Properties* dialog box.

In the *Update Cycles* tab of the *Project Properties* dialog box, the user-defined cycles 1 - 5 are offered for the project-related definition at the end of the default update cycles list. Only these user cycles can be parameterized.

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

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

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

For the update cycles, the following differentiations are made:

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

**Note:**

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

For the application of the update cycles, the following settings are recommended depending on the type of cycle selected:

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

This dialog is displayed if a Smart Object $\rightarrow$ I/O Field is configured. This dialog can also be initiated via $\rightarrow$ on the corresponding object.
Dynamic Wizard

This page is displayed by D on the entry Make Property Dynamic in the Standard Dynamicstab of the Dynamic Wizard.
Tag Connection for Object Property

This menu is displayed by selecting the column Current of an object property that has been made dynamic through a tag with a \( \downarrow \) R.
Dynamic Dialog

If you select the trigger button when in a dynamic dialog box, the dialog box for changing the update cycle is selected.
C action for a Property

If you select the trigger button in the editor while configuring a C-Action, the dialog box for changing the update cycle is selected.

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

Change of the picture cycle.
Window Cycle

Change of the window cycle.
3.3.1.5 Execution of Background Scripts (Global Script)

- The execution of background scripts (Global Script) is dependent on various variables depending on the configuration:
- Time trigger (cyclic execution, exception: acyclic = one time)
- Time cycle
- Time
- Event trigger

or only once

![Image of trigger configuration](image-url)
Time Cycle

- The configured time factor of the global action defines when the defined sequence of actions must be processed. In addition to already described default cycle and the associated time settings of 250 ms to 1 h (or user-defined cycle 1 - 5), the time triggers:
  - Hourly (minutes and seconds)
  - Daily (hours, minutes, seconds)
  - Weekly (day of the week, hours, minutes, seconds)
  - Monthly (days, hours, minutes, seconds)
  - Yearly (months, days, hours, minutes, seconds)
can be selected.
Tag Trigger

If the action is activated dependent on one or more tags, the event trigger must be set as a tag trigger. You do this in the same way as for the tag trigger for object properties.
The 2 second cycle is set by default. The configurator, however, can set the following time factors instead of the default value:

At the start and at the end of the set time frame, the values of the tags selected are determined. If the value of at least one of the tags has changed, the trigger for the global action is tripped.

Note the high system loads when triggering upon change. This setting is not always appropriate. The same advice applies here as does for the Update object.

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

Definitions

The term adding dynamics means the changing of stati (e.g. position, color, text, etc.) and the reaction to events (e.g. mouse click, keyboard operation, value change, etc.) during runtime.

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

3.3.2.1 Making Properties dynamic

The properties of an object (position, color, text, etc.) can be set statically and be changed dynamically in runtime. All properties with a light bulb in the Dynamic column can be made dynamic. Once a property has been made dynamic, a colored symbol, which depends on the type of dynamic, is displayed instead of the white light bulb. Topics (e.g. Geometry) that have been made dynamic are shown in bold type.
### 3.3.2.2 Making Events dynamic

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

![Object Properties](image)

### 3.3.2.3 Dynamization Types for Objects

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

<table>
<thead>
<tr>
<th>Type of Dynamic</th>
<th>A</th>
<th>B</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Wizard</td>
<td>x</td>
<td>x</td>
<td>Standard prompted method when configuring</td>
<td>Only for certain types of dynamization. Always generates a C-Action!</td>
</tr>
<tr>
<td>Direct Connection</td>
<td>x</td>
<td></td>
<td>The fastest method for making a picture dynamic, greatest performance in runtime</td>
<td>Restricted to one connection and can only be used in a picture.</td>
</tr>
<tr>
<td>Tag Connection</td>
<td></td>
<td>x</td>
<td>Easy to configure</td>
<td>Limited options for dynamization</td>
</tr>
<tr>
<td>Dynamic Dialog</td>
<td>x</td>
<td></td>
<td>Quick and clear; for value ranges or a number of alternatives; high performance in runtime</td>
<td>Cannot be used for all types of dynamization.</td>
</tr>
<tr>
<td>C-Action</td>
<td>x</td>
<td>x</td>
<td>Almost unlimited possibilities for dynamization thanks to the powerful script language (ANSI-C)</td>
<td>Possibility of errors due to wrong C-instructions slower performance compared with other types of dynamization, therefore always check whether or not you can achieve your aim through a different type of dynamization.</td>
</tr>
</tbody>
</table>

**Legend:**

A: Dynamization of the Object Property  
B: Dynamization of the Object Event
# Opening the Dialog Boxes for the Dynamization

<table>
<thead>
<tr>
<th>Dialog Box</th>
<th>Open Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Dialog</td>
<td>Not all objects have such a dialog box. Automatic when creating these objects. Select object in picture press and hold the SHIFT key D on object. Select object in picture R on object to open its pop-up menu select configuration dialog</td>
</tr>
<tr>
<td>Dynamic Wizard</td>
<td>Select object in picture select property or event select Wizard D on Wizard to start it. The Wizard must be check-marked in View Toolbars...</td>
</tr>
<tr>
<td>Direct Connection</td>
<td>Select object in picture display object properties select events tab R on the action column to open the pop-up menu select direct connection.</td>
</tr>
<tr>
<td>Tag Connection</td>
<td>Select object in picture display object properties select properties tab R on the dynamic column to open the pop-up menu select tag in the following dialog box, select and apply the appropriate tag.</td>
</tr>
<tr>
<td>Dynamic Dialog</td>
<td>Select object in picture display object properties select properties tab R on the dynamic column to open the pop-up menu select dynamic dialog in the following dialog box, configure and apply the appropriate dynamic</td>
</tr>
<tr>
<td>C-Action</td>
<td>Select object in picture display object properties select properties or events tab R on the dynamic or action column to open the pop-up menu select C-Action configure and compile the appropriate C-Action.</td>
</tr>
</tbody>
</table>
## Results and Presentation

<table>
<thead>
<tr>
<th>Dialog Box</th>
<th>Result</th>
<th>Presentation (Symbol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Wizard</td>
<td>A C-Action is always generated.</td>
<td>Green lightning bolt</td>
</tr>
<tr>
<td>Direct Connection</td>
<td></td>
<td>Blue lightning bolt</td>
</tr>
<tr>
<td>Tag Connection</td>
<td></td>
<td>Green light bulb</td>
</tr>
</tbody>
</table>
| Dynamic Dialog   | Automatically generated C-Action (InProc), this C-Action can subsequently be expanded, but the performance advantage will be lost in the process. | Red lightning bolt  
Upon change in C-Action, switch to Green lightning bolt |
| C-Action         | Configured C-Script                                                    | Green lightning bolt  
Yellow lightning bolt - the action still has to be compiled                      |
3.3.3 WinCC System Environment

By default, WinCC is installed in the C:\Siemens\WinCC\ folder. You can change this default path during installation.

3.3.3.1 Folder Structure of the WinCC System

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

**Files in the WinCC Default Folder**

In the default WinCC path, the following folders and files are important for the configurator or commissioning engineer:

<table>
<thead>
<tr>
<th>Folder</th>
<th>File Name, Extension</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagnostics</td>
<td>License.log</td>
<td>Current log entries regarding the license checks and/or violations.</td>
</tr>
<tr>
<td></td>
<td>License.bak</td>
<td>The log file with the license information from the last startup.</td>
</tr>
<tr>
<td></td>
<td>WinCC_Op_01.log</td>
<td>Operator messages generated by WinCC during runtime.</td>
</tr>
<tr>
<td></td>
<td>WinCC_Sstart_01.log</td>
<td>System messages generated by WinCC on startup. An important file when troubleshooting. The file contains messages about missing tags and wrongly executed scripts.</td>
</tr>
<tr>
<td></td>
<td>WinCC_Sys_01.log</td>
<td>System messages generated by WinCC during runtime. An important file when troubleshooting. The file contains messages about missing tags and wrongly executed scripts.</td>
</tr>
<tr>
<td></td>
<td>S7chn01.log</td>
<td>System message of the channel used (in this case S7)</td>
</tr>
<tr>
<td>aplib</td>
<td>Library path</td>
<td>The header files, all standard functions and all internal functions are stored in subfolders.</td>
</tr>
</tbody>
</table>
Files in the WinCC Default Folder

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

<table>
<thead>
<tr>
<th>Folder</th>
<th>Subfolder, File Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>aplib</td>
<td>library.pxl</td>
<td>Symbols of the default library of WinCC.</td>
</tr>
<tr>
<td></td>
<td>Report, Wincc, Windows</td>
<td>Folders for standard functions; they can be adjusted at any time.</td>
</tr>
<tr>
<td></td>
<td>Allocate, C_bib, Graphics, Tag</td>
<td>Folders for internal functions; they cannot be adjusted.</td>
</tr>
<tr>
<td>syslay</td>
<td></td>
<td>All print layouts that are automatically copied by WinCC to the folder <code>prt</code> in the project path during the creation of a project.</td>
</tr>
<tr>
<td>wscripts</td>
<td>Dynwiz.cwd</td>
<td>Dynamic Wizard of the Graphics Designer. You can create your own scripts at any time. These scripts are given the extension <code>.wnf</code>.</td>
</tr>
<tr>
<td></td>
<td>wscripts.deu</td>
<td>This path contains the script files for German. This path depends on the language installed.</td>
</tr>
<tr>
<td></td>
<td>Wscripts.enu</td>
<td>This path contains the script files for English. Since English is the default language, this path is always created.</td>
</tr>
<tr>
<td></td>
<td>Wscripts.fra</td>
<td>This path contains the script files for French. This path depends on the language installed.</td>
</tr>
</tbody>
</table>
### Files in the Default WinCC Folder

The following application programs are stored in the following folders during the installation of WinCC:

<table>
<thead>
<tr>
<th>Folder/File</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\sqlany\isql.exe</code></td>
<td>Interactive program used for looking at the data in the database of a WinCC project.</td>
</tr>
<tr>
<td><code>\bin\Wunload.exe</code></td>
<td>Wizard used for emptying the online tables in the database of the WinCC project, e.g. removing the messages and measured value data stored. The wizard automatically provides the runtime tables for unloading; additional tables can be added or removed from the list by the user at any time. This tool must be used in the offline status of a WinCC project. It cannot be used in runtime mode. Messages and measured values can be exported during runtime via the optional STORAGE package.</td>
</tr>
<tr>
<td><code>\bin\Wrebuild.exe</code></td>
<td>Wizard for reconstructing the database; it cannot be used in runtime mode!</td>
</tr>
<tr>
<td><code>\SmartTools\CC_GraficTools\metaVw.exe</code></td>
<td>Viewer for graphics files (e.g. print jobs, exported symbols) in EMF format (extended meta file).</td>
</tr>
<tr>
<td><code>\SmartTools\CC_GraficTools\wmfcode.exe</code></td>
<td>Viewer for graphics files in WMF format (windows meta file).</td>
</tr>
<tr>
<td><code>\SmartTools\CC_OCX_REG\ocxreg.exe</code></td>
<td>For registering or canceling the registration of additional OLE Control components (OCX).</td>
</tr>
<tr>
<td><code>\SmartTools\CC_OCX_REG\Regsvr32.exe</code></td>
<td>Is called by ocxreg.exe.</td>
</tr>
</tbody>
</table>
3.3.4 WinCC Project Environment

Note:
Create a special project folder for WinCC projects, e.g. WinCC_Projects. In this way, you keep the WinCC system and the configured data totally separate from each other, simplifying the task of backing up data. You also avoid the danger of losing data (due to operator errors) if you need to deinstall WinCC.

3.3.4.1 WinCC Project - Folder Structure

A project in WinCC consists of a complete folder structure with corresponding content. After creating a new project in the WinCC Explorer (via the menu entry File -> New), a new folder structure is established as follows:

<table>
<thead>
<tr>
<th>WinCC as Standard</th>
<th>WinCC with Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Folder Structure Diagram" /></td>
<td><img src="image" alt="Folder Structure Diagram" /></td>
</tr>
</tbody>
</table>
## Contents of the Project Folders

<table>
<thead>
<tr>
<th>Folder</th>
<th>Extension</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Path</td>
<td>.db</td>
<td>The database with the configuration data.</td>
</tr>
<tr>
<td></td>
<td>rt.db</td>
<td>The database with the runtime data, measured values and messages.</td>
</tr>
<tr>
<td></td>
<td>.mcp (master control program)</td>
<td>Main file of the WinCC project. The project is opened with this file.</td>
</tr>
<tr>
<td></td>
<td>.pin</td>
<td>Project.pin</td>
</tr>
<tr>
<td>GraCS</td>
<td>.pdl (picture design language)</td>
<td>The configured pictures.</td>
</tr>
<tr>
<td></td>
<td>.sav</td>
<td>Backup files of the picture files with the last configuration state.</td>
</tr>
<tr>
<td></td>
<td>.gif (bitmap), .wmf (windows meta file), .emf (extended meta file)</td>
<td>Picture files</td>
</tr>
<tr>
<td></td>
<td>.act (action)</td>
<td>Exported C-Actions</td>
</tr>
<tr>
<td></td>
<td>.pdd</td>
<td>Default.pdd Setting parameters for the graphic editor (default settings of the individual objects in the Object Palette)</td>
</tr>
<tr>
<td>Library</td>
<td>.h (header)</td>
<td>Ap_pbib.h (Project function declarations)</td>
</tr>
<tr>
<td></td>
<td>.pxl</td>
<td>Library.pxl (Project symbol library)</td>
</tr>
<tr>
<td></td>
<td>.fct</td>
<td>Project functions</td>
</tr>
<tr>
<td></td>
<td>.dll (dynamic link library)</td>
<td>Separate function libraries that have been created in a C-development environment.</td>
</tr>
<tr>
<td>Meld</td>
<td>.pas (action definition)</td>
<td>Project actions that run as background actions dependent on the set trigger.</td>
</tr>
<tr>
<td>Pass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pde</td>
<td>.rpl (report picture language)</td>
<td>Page layouts for print jobs. All the predefined WinCC default layouts begin with @. All system variables (including tags) are identified by this prefix.</td>
</tr>
<tr>
<td>Prt</td>
<td>.rpl (report picture language)</td>
<td>Page layouts for print jobs. All the predefined WinCC default layouts begin with @. All system variables (including tags) are identified by this prefix.</td>
</tr>
<tr>
<td>Computer Name, e.g. Zip-ws1</td>
<td>\GraCS\GraCS.ini</td>
<td>Initialization file for the graphic editor.</td>
</tr>
</tbody>
</table>
### Optional: Files which can be created during the Configuration

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

Requirement

The HMI system (WinCC) in the plant should start automatically when Windows is started. The HMI station operator does not need to know anything about using Windows (e.g. activating WinCC under Windows NT).

Solution

WinCC is started automatically during the PCs startup routine. This function is set in the Startup folder of Windows.

Create a Connection

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure in NT 4.0:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Windows Explorer, go to the WinNT\Profiles\All Users\Start Menu\Programs\Startup folder. WinNT is the folder in which Windows NT has been installed.</td>
</tr>
<tr>
<td>2</td>
<td>In this folder, create a new connection by (\Box R \rightarrow New \rightarrow Connection).</td>
</tr>
<tr>
<td>3</td>
<td>Establish the connection to the program mcp.exe (Master Control Program) in the \bin folder of WinCC.</td>
</tr>
<tr>
<td>4</td>
<td>Give the connection a name.</td>
</tr>
</tbody>
</table>

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

Note:

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

The operator now sees the familiar start screen of the system. To prevent an accidental or intentional switch to the configuration (running in the background) or Windows applications by the operator, appropriate measures must be taken. Operators must also not be able to open the database runtime window, since they can close the WinCC database connection from there.
No Selection of:
Operators should have no way of switching from WinCC runtime to:
• the WinCC Explorer of WinCC (configuration environment)
• the runtime window of the SQL database of WinCC (Sybase SQL Anywhere), since they could use this route to terminate the WinCC database connection. WinCC can then no longer operate
• the task bar of Windows, since it can be used to start all installed programs
• the current task window, since the application can be closed from there

Settings required on the Computer
The following key combinations must be locked to prevent the operators from performing these operations.

Key combinations are locked in the WinCC Explorer from the Computer Properties dialog box.
The exact definition of the individual key combinations can be obtained from the WinCC help or from the operating system help.
Settings required for Runtime

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

The locks are set in the WinCC Explorer from the Computer Properties dialog box. The exact definition of the individual key combinations can be obtained from the WinCC help or from the operating system help.

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

Note:
If the keys mentioned above are all or partly locked, access to the configuration must be provided for the configurator or service personal using a key especially configured for this purpose. This also applies to the proper shut down of the system.

These functions must not be readily accessible. Also add access protection to the button by setting the password property.
3.3.6 Coordinated Shut Down of WinCC

Keeping the System Consistent

To maintain the consistency of the system, it is very important to exit WinCC properly. WinCC runtime should always be exited first using the intended button. Afterwards, close the project from the WinCC Explorer.

If this procedure is not adhered to, data losses can occur, the database be damaged or - in the worst case - the system crash.

Controlled Exit

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

Power Failure

To avoid data losses due to fluctuating currents or a power failure, a detailed data backup concept for the HMI system should be devised and put into place.

In any case, a UPS (uninterruptible power supply) should be provided for the WinCC station. This can be implemented by connecting the station to the plant's own UPS or by connecting a separate UPS to the server of WinCC. This applies to both single-user and multi-user systems, regardless of the operating system being used (Windows NT).

The UPS used must also include special shutdown software for Windows NT, which - in the event of a power failure and shutdown after a specified time interval - automatically exits the operating system and all active applications without the loss of data; e.g. APC UPS 600 with power shutdown software for Windows NT.

3.3.6.1 Notes on how to Install a UPS

The WinCC station must have a serial interface to which a UPS with appropriate test software can be connected. If there is no a serial interface free on the station (e.g. they are all occupied by a printer or PLC), you must install an additional interface card.

Serial interfaces that are used for connecting two or more peripherals (e.g. via switches) are not supported by the majority of UPS systems and are certainly not recommended, since the system needs to be monitored constantly.

An appropriate monitoring service is installed in the operating system. This monitoring service then has to be assigned shutdown parameters, so that coordinated and orderly exiting of the system is guaranteed. The shutdown process for application software must be activated under all circumstances, so that WinCC is closed without any loss of data in the event of a shutdown. You must choose a Save Time before shutdown, that is sufficiently long in order for all active applications to be exited properly. Most software packages for UPS systems also offer a time-controlled Shutdown, e.g. for the weekend or night time. This function can be used to achieve a deliberate shut down of the WinCC system without any operator input.
3.3.7 Data Backup

When should Data to be Backed Up?

- A WinCC project must be repeatedly saved and backed up for the following reasons:
  - During the configuration phase.
  - Before exporting/importing data (e.g. when importing tags, multilingual picture texts, message texts and multilingual message texts).
  - Before forming or unloading the WinCC database.
  - Before editing the database using tools such as interactive SQL access.
  - The configuration data must be backed up for installation on the destination system at the end user.
  - Before taking over data for a similarly structured project.

What Media are suitable?

<table>
<thead>
<tr>
<th>Medium</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floppy Disks</td>
<td>Can be read by almost any system.</td>
<td>Insufficient capacity (even when data is compressed).</td>
</tr>
<tr>
<td>ZIP Disks</td>
<td>Inexpensive, adequate capacity, direct and quick access possible via Windows; simple to install; portable, for use in the plant.</td>
<td></td>
</tr>
<tr>
<td>Streamers (e.g. on the network)</td>
<td>Automatic backup possible (daily), very high capacity.</td>
<td>Mostly available only in office environments, no direct access to the data due to the special format being used for saving.</td>
</tr>
<tr>
<td>Hard Disk on another PC (e.g. Laplink)</td>
<td>No need to deal with other media, data can be used directly.</td>
<td>Slow, unsuitable for large volumes of data.</td>
</tr>
<tr>
<td>MOD</td>
<td>High level of data integrity, reusable, backup of messages and measured values possible in runtime mode.</td>
<td>A special drive is required for reading and writing.</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>High capacity, can be read by almost any system, suitable for long-term archiving.</td>
<td>A special drive is required for writing, medium not reusable.</td>
</tr>
</tbody>
</table>
Cleaning the Project before the Data Backup

To clean the project and keep it as small as possible before the data backup for the end customer or transfer of the project data, the following data can be deleted or cleaned using other programs.

- All backup files in the \GraCS\*.sav folder of the project.

If you have not created any layouts of your own (Report Designer) for the documentation, the system layouts can be deleted from the \Prt folder. When a new project is created, all system layouts are automatically copied from the \Siemens\WinCC\syslay folder to the project folder.

What Data must be Backed Up?

If only the data of a WinCC project is to be backed up, the following individual files and folders with all the files contained in them must be backed up.

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

If you have created project-wide components (standard functions, objects in the project library), the following files must also be backed up from

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

This data is not generated when WinCC is reinstalled.
3.3.8 Copying a Backed Up WinCC Project to a new Destination Computer

Installing the System Software

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

Supplements

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

Windows Software

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

OCX, ActiveX

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

Network

If the project has been configured for a multi-user system, the network must be fully installed on the WinCC destination computer before the WinCC data is copied. Make a note of the necessary computer names in the configured computer environment, since you will need them when assigning the parameters of the project copied. The computer name is also required as a parameter for a single-user system. Therefore, you must know the name of the destination computer or determine its name via the Windows Control Panel.
Copying the Data and Starting the Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure when Transferring Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project folder (e.g. WinCC_Projects).</td>
</tr>
<tr>
<td>2</td>
<td>Copy the entire backed up path to this project folder as a subfolder (e.g. \WinCC_Projects\Varia_00). The name of the WinCC project folder can be changed if desired. When renaming the files in the project folder (varia_00.mcp, varia_00.db, varia_00.pin, varia_00.log), make sure that all the files are given the same name (with the exception of the extension).</td>
</tr>
<tr>
<td>3</td>
<td>Open the project in the WinCC Explorer.</td>
</tr>
<tr>
<td>4</td>
<td>If necessary, change the project-specific settings. If the type has to be changed, additional adjustments must also be made to the computer properties. These adjustments are described in the WinCC Help.</td>
</tr>
<tr>
<td>5</td>
<td>Check the Computer Name and, if necessary, change it at the Computer Properties in the General Information tab. If the computer name in the WinCC project does not match the computer name of the destination system, an error message is displayed if you activate the project, i.e. when you activate runtime.</td>
</tr>
</tbody>
</table>
| 6    | - If you have used your own standard functions (*.fct) in the project, the backed up standard functions must be copied to the default WinCC path \Siemens\WinCC\aplib and then be made known in the function tree:  
  - With the project open, start the Global Script editor.  
    Regenerate the declaration structure using the Options menu entries.  
    The new functions are now visible in the function tree.  
    Regenerate Header menu entries.  
  - Activate the project and check for a proper start up. |
3.3.9 Reuse - Transfer of Project Parts to a New or Existing Project

Reasons for reusing Data

A WinCC project can be generated in different ways. The most important aspects concern the reusability of already existing project parts from similar projects or the transfer of data from pre-configured sample projects.

Project Team

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

Data in the Database

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

Note:

Note that every change made in the database influences the structure and the access of the database. Lots of unnecessary changes (possibly with deletions) can lead to the WinCC database no longer being optimally configured and consequently to a loss of performance.

Single-User System

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

Multi-User System or Multi-Client

In contrast, if a project is being configured on a multi-user system or on a multi-client, changes can be made simultaneously to the configuration in different areas of the database. For example, one person configuring the system can edit Alarm Logging while another 'system configurer' edits the archiving system (measured-value acquisition).
Changing the Project Type: Conversion of Single-User and Multi-User Systems

Any project can be converted from one WinCC configuration system to another one. If a change of configuration between the configuration station(s) and the customer’s system is planned beforehand, no elements specific to WinCC, that are oriented to a multi-user system, may be used. For example, no internal tags can be used on the local computer if the system being configured is actually for a single-user system at the customer.

When creating a new project (regardless whether it is for a single-user or multi-user system), you must know beforehand whether or not the project is to be based on an existing basic project with preconfigured Alarm Logging and/or archiving data. This data stored in the database can only be transferred to other projects through reconfiguring or, if possible, through export - import.

3.3.9.1 Transfer of Pictures

Configured pictures can be transferred at any time. You can copy the picture files (*.pdl) directly from the source folder to the target folder of the WinCC project path \GraCS using the Windows Explorer (useful if you have to copy several picture files).

Below is an extract from the project for the picture configuration:

You can also transfer pictures by opening a picture file (picture.pdl) in the Graphics Designer via the menus File ➔ Open. Afterwards, the picture is saved in the current picture folder (\GraCS) via the menus File ➔ Save As. This procedure is suitable if the picture files are being used as a basis and will be customized immediately.
References in Pictures

- Structures from the data type area
- Internal or external tags
- System tags
- Picture objects that are stored as a bit map or meta file (e.g. for status displays or graphic objects)
- Other pictures that are used as graphic or process boxes or as displayed windows
- Project functions used

Access rights

The following references must also be defined:

- Definition of the structures under the data types, e.g. controller or template structures for customized objects
- Definition of the communication channels and logical connections with the definition of the tags (possibly with tag groups)
- Definition of the internal tags or system tag names (starting with @)
- Transfer of picture elements (*.gif or *.emf) by copying them from the \GraCS folder
- Transfer of picture window contents by copying additional picture files (*.pdl) from the \GraCS folder
- The project functions used must be copied from the source project to the \Library folder of the new project. In addition, these functions must be stored in the function tree of the Global Script editor via the Regenerate Header menu. This procedure has been detailed in the chapter Development Environment for C-Scripts.

Definition of the access rights to be used in the User Administrator editor. The access rights used (e.g. for control buttons) must be defined for the group specifications.
3.3.9.2 Transfer of Symbols and Bitmaps

Copying

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

Picture browser for the graphic object:
Importing

Symbols can either be integrated into a picture using the method just described or can be copied directly into the graphic picture being edited via the Insert → Import menus. In the latter case, you do not have to copy a file, instead you import the symbol you want directly by accessing the path of the source project (\GraCS) and then selecting the desired symbol file (*.bmp, *.emf, *.wmf). After being imported, the symbol is then immediately displayed as an object in the picture (top left).

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

3.3.9.3 Transferring a Project Library (with preconfigured Symbols and Customized Objects)

Global Library

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

The preconfigured blocks can now continue to be used at any time in the new project:

![Project Library Image]

**Note:**

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Transferring Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the library.</td>
</tr>
<tr>
<td>2</td>
<td>Select the desired symbol using the and drag the symbol into the picture while holding down the mouse button (Drag and Drop).</td>
</tr>
<tr>
<td>3</td>
<td>Via the File → Export... menus, open the dialog box for saving the symbol.</td>
</tr>
<tr>
<td>4</td>
<td>Save the symbol.</td>
</tr>
</tbody>
</table>

New Project Library

These exported symbols are now available as separate symbol files and can be used individually by being imported. If these symbols are used frequently in the project, they should be integrated again into the new project library. You do this by calling the symbol library, in particular the project library.

Create your own symbol folder, e.g. with the aid of the folder icon in the toolbar of the library window, and copy the imported symbols to this folder via Drag and Drop. In this way, you can transfer some of your symbols from projects and add additional specific symbols to create a new project-specific library again.
3.3.9.4 Transfer of Actions

Actions that are frequently required in a project or that are to be copied from one object action to another object action are stored as separate files. These files are stored in the \GraCS folder with the extension .act. They can be transferred at any time by being copied from the source folder to the target folder.

An action file is stored into the target file named by the user (with the extension .act for action) via the toolbar button Export Action from the C-Actions editor.

A stored action file is transferred to an object action in the picture of the new project via the Import Action toolbar button. Please also note the description in the chapter Development Environment for C-Scripts.

Note:
More often used actions can also be defined as project or standard functions.
3.3.9.5 Transfer of Tags

- The tag management of WinCC can be supplemented in a number of different ways:
- Reading S5 data tags or S7 data tags using Wizards (Dynamic Wizard)
- Transfer of S7 tags via the PCS7 Mapper
- By importing and exporting text lists using the program Var_Exim
- Interactive access to the database tables (tag tables)

Specifically programmed Dynamic Wizards or programs that utilize the WinCC API functions to generate new data in Tag Management

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

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

Transferring S5/S7 Data Tags using the Dynamic Wizard

The data area definitions generated using STEP5/STEP7 software can be read into the Tag Management of WinCC with the aid of the Dynamic Wizard. The following steps must be performed:

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Transferring S5 or S7 Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Back up the data of the project. Changes are made in the database.</td>
</tr>
<tr>
<td>2</td>
<td>Export the assignment list using the STEP software. A file named prj_zuli.SEQ is created.</td>
</tr>
<tr>
<td>3</td>
<td>Remove any special symbols (e.g. for program calls) that are not required for import into WinCC from this exported file. You can do this with a typical text editor, such as Wordpad. The assignment list must not contain any blank lines.</td>
</tr>
<tr>
<td>4</td>
<td>Open the target project in the WinCC Explorer. The project must be in configuration mode (runtime not active).</td>
</tr>
<tr>
<td>5</td>
<td>Open the Graphics Designer editor. In any picture, go to the Dynamic Wizard (is displayed via View Toolbars...) and select the Import Functions tab. From there, select the function Import S7 - S5 Assignment List. Then the source file (.seq) with the path must be specified (using the button).</td>
</tr>
</tbody>
</table>

Also specify the logical connection, in which the tag descriptions of the assignment list are to be placed.

The data is now entered into the WinCC tag management. The tag names used in a WinCC project must be unique. The tags are added to the (possibly) already existing WinCC Tag Management. The tag name is the key to this.
**Transferring Tags using the Help Program**

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

**Note:**

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

The tags defined in Tag Management can be **exported** at any time as a text file to supplement the list of tags. Afterwards, this data generated must be **imported** back into the tag management of the project. The files created are in CSV (comma separated values) format and can be read and further processed using any editing program.

- For this purpose, a separate application program is available on the WinCC CD-ROM in the \SmartTools\CC_VariableImportExport folder. This Windows program is provided as a help application for:
  - the export of the Tag Management data
  - the import of tag data that has already been generated externally
  - the mass data configuration

The following steps have to be performed now to export or import the data:
<table>
<thead>
<tr>
<th>Step</th>
<th><strong>Procedure: Importing / Exporting Tags</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open your WinCC project in the <em>WinCC Explorer</em>.</td>
</tr>
<tr>
<td>2</td>
<td>Define connections (channel-DLL - logical connection - connections parameters) that are currently not available but will be later required for importing. Only do this, however, in the new project. This may require a second export-import procedure.</td>
</tr>
<tr>
<td>3</td>
<td>Activate the <em>Var_exim</em> program via a . The user interface of this program is displayed below.</td>
</tr>
</tbody>
</table>

![VARIABLE EXPORT / IMPORT](image)

- Path: \\TESTAUTOMATION\WinCC50_Project pivotal\pictu_00
- Filename:  
- List separator: Get from Regional Settings
- Show: var con dex diag Ready Execute End
### Import - Export

For the export or import, the following steps have to be performed:

<table>
<thead>
<tr>
<th>Location, Action</th>
<th>Import</th>
<th>Export</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Path</strong></td>
<td>Select the project folder containing the files for the tag import. The selection is made by selecting the file named .mcp. The files containing the data for the import must be located in the same folder as the project file.</td>
<td>Specify the project folder for the tag export. The selection is made by selecting the file named .mcp.</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Select <strong>Import</strong>. If already existing tag data is to be overwritten, select <strong>Import Overwrite</strong>.</td>
<td>Select <strong>Export</strong>.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Click on execute. The dialog box then displayed shows the parameters set and executes the procedure after you have clicked OK. Due to the checks performed, import takes more time.</td>
<td>Click on execute. The dialog box then displayed shows the parameters set and executes the procedure after you have clicked OK.</td>
</tr>
<tr>
<td><strong>Status Display</strong></td>
<td>End Export/Import</td>
<td>End Export/Import</td>
</tr>
<tr>
<td><strong>Tag File</strong></td>
<td><strong>Name_vex.csv</strong>&lt;br&gt;Basis for import: consists of a header and data records.</td>
<td>The tag list generated is stored in this file as text. This file can be opened by clicking on the var button or be edited using a text editor (Notepad) or EXCEL.</td>
</tr>
<tr>
<td><strong>Tag File</strong></td>
<td><strong>Name_cex.csv</strong>&lt;br&gt;Basis for import: consists of a header and data records (structural components).</td>
<td>This file contains the currently configured connections which are referred to in the tag file. This file can be opened by clicking on the con button or be edited using a text editor (Notepad) or EXCEL.</td>
</tr>
<tr>
<td><strong>Data Structure File</strong></td>
<td><strong>Name_dex.csv</strong>&lt;br&gt;Basis for import: consists of a header and data records.</td>
<td>If tags with data structure types are included, this file containing structural information is also generated. You can edit its contents using a text editor (Notepad) or EXCEL.</td>
</tr>
<tr>
<td><strong>Diagnostics File</strong></td>
<td><strong>Diag.txt</strong>&lt;br&gt;A diagnostics file containing information as to which tags were not able to be imported.</td>
<td></td>
</tr>
</tbody>
</table>

This program can be exited by clicking on the *End* button.
Tag Lists

The following table describes the structure of the tag lists.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Name</td>
<td>char</td>
<td>Tag name</td>
</tr>
<tr>
<td>Conn</td>
<td>char</td>
<td>Connection</td>
</tr>
<tr>
<td>Group</td>
<td>char</td>
<td>Group name</td>
</tr>
<tr>
<td>Spec</td>
<td>char</td>
<td>Internal tag or address (matching the connection type)</td>
</tr>
<tr>
<td>Flag</td>
<td>DWORD</td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td>DWORD</td>
<td></td>
</tr>
<tr>
<td>CType</td>
<td>DWORD</td>
<td>Tag type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 BIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 SBYTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 BYTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 SWORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 WORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 SDWORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 DWORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 TEXT_8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 TEXT_16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 Raw Data Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 Field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 Structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 BITFIELD_8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 BITFIELD_16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 BITFIELD_32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 Text Reference</td>
</tr>
<tr>
<td>CLen</td>
<td>DWORD</td>
<td>Length of the tags</td>
</tr>
<tr>
<td>CPro</td>
<td>DWORD</td>
<td>Internal or external tag</td>
</tr>
<tr>
<td>CFor</td>
<td>DWORD</td>
<td>Format conversion</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>BOOL</td>
<td>Error at upper limit</td>
</tr>
<tr>
<td>P2</td>
<td>BOOL</td>
<td>Error at lower limit</td>
</tr>
<tr>
<td>P3</td>
<td>BOOL</td>
<td>Conversion error</td>
</tr>
<tr>
<td>P4</td>
<td>BOOL</td>
<td>Write error</td>
</tr>
<tr>
<td>P5</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>BOOL</td>
<td>Substitute value on upper limit error</td>
</tr>
<tr>
<td>L2</td>
<td>BOOL</td>
<td>Substitute value on lower limit error</td>
</tr>
<tr>
<td>L3</td>
<td>BOOL</td>
<td>Start value</td>
</tr>
<tr>
<td>L4</td>
<td>BOOL</td>
<td>Substitute value on connection error</td>
</tr>
<tr>
<td>L5</td>
<td>BOOL</td>
<td>Upper limit valid</td>
</tr>
</tbody>
</table>
### Field, Type, Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L6</td>
<td>BOOL</td>
<td>Lower limit valid</td>
</tr>
<tr>
<td>L7</td>
<td>BOOL</td>
<td>Start value valid</td>
</tr>
<tr>
<td>L8</td>
<td>BOOL</td>
<td>Substitute value valid</td>
</tr>
<tr>
<td>LF1</td>
<td>double</td>
<td>Upper limit</td>
</tr>
<tr>
<td>LF2</td>
<td>double</td>
<td>Lower limit</td>
</tr>
<tr>
<td>LF3</td>
<td>double</td>
<td>Start value</td>
</tr>
<tr>
<td>LF4</td>
<td>double</td>
<td>Substitute value</td>
</tr>
</tbody>
</table>

### Scaling

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF</td>
<td>DWORD</td>
<td>1 when scaling is defined</td>
</tr>
<tr>
<td>SPU</td>
<td>double</td>
<td>Range of values, process from</td>
</tr>
<tr>
<td>SPO</td>
<td>double</td>
<td>Range of values, process up to</td>
</tr>
<tr>
<td>SVU</td>
<td>double</td>
<td>Range of values, tag from</td>
</tr>
<tr>
<td>SVO</td>
<td>double</td>
<td>Range of values, tag up to</td>
</tr>
</tbody>
</table>

### Connection List

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConName</td>
<td>char</td>
<td>Logical connection name</td>
</tr>
<tr>
<td>Unit</td>
<td>char</td>
<td>Channel unit</td>
</tr>
<tr>
<td>Common</td>
<td>char</td>
<td>General</td>
</tr>
<tr>
<td>Specific</td>
<td>char</td>
<td>Specific connection parameters</td>
</tr>
<tr>
<td>Flag</td>
<td>DWORD</td>
<td></td>
</tr>
</tbody>
</table>

### Data Structure List

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStructure</td>
<td>short</td>
<td>Data structure name or component name</td>
</tr>
<tr>
<td>Type ID</td>
<td>short</td>
<td>Identification (is used in tag list under CType)</td>
</tr>
<tr>
<td>Creator ID</td>
<td>short</td>
<td></td>
</tr>
</tbody>
</table>

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

The tag data in the text list is adapted using the following special instructions:

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

**Before** you initiate the import of the edited or new tags, first perform a **data backup** of your project, since changes will be made to the database. These database changes cannot be reversed in WinCC.
3.3.9.6 Transfer of Multilingual Texts (from Pictures, in Messages)

Multilingual Texts of Pictures

The multilingual texts stored in the pictures can be transferred to the new project by copying the picture itself.
The respective language must be added to the text library in the new project. Check the settings in the Text Library. Each language must have its own column!

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

Transfer of Pictures with Text References

- If text references are used in the transferred pictures, the following data must also be transferred with it:
- The associated tags (export or redefinition) from the tag management of the WinCC project
- The texts from the text library
- The text reference tags must be provided with the valid text identification numbers (text IDs). Check whether the text IDs still agree with the associated texts.
Transfer of Texts from the Text Library

If only some of the texts from the text library are transferred, you must adjust the text IDs accordingly. Texts from the text library can be transferred via the export/import mechanism of the Text Library editor.

3.3.9.7 Transfer of Messages

- The information basis for messages (alarms) requires
- modification and
- a great deal of configuration work due to the volume (mass data).

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

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

Transfer of Messages from COROS

The procedure for the sources quoted above looks as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Transfer of existing COROS Message Texts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In COROS, export the message information (mltdtexte.txt)</td>
</tr>
<tr>
<td>2</td>
<td>In WinCC, import this message file using the Dynamic Wizard Import Functions ➔ Import Messages. The data is now imported into the current WinCC project.</td>
</tr>
</tbody>
</table>

Transfer of Messages from a WinCC Project

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Transfer of existing WinCC Message Texts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the current project, open the Alarm Logging editor.</td>
</tr>
<tr>
<td>2</td>
<td>Initiate the export of the messages by selecting the Messages ➔ Export Single Messages menus.</td>
</tr>
<tr>
<td>3</td>
<td>Specify the destination text file in which the message information to be exported is to be stored. At Selection, select the messages to be exported using the criteria, e.g. Number, Message Class.</td>
</tr>
<tr>
<td>4</td>
<td>Start the export procedure by clicking on Export. A text file containing items of message information separated by commas is now created.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Transfer of existing WinCC Message Texts</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>5</td>
<td>Close the current project and open the new one. Open the Alarm Logging again and define the necessary message classes and message types. Define one message per message type, in order to retain the basic framework for the following import.</td>
</tr>
<tr>
<td>6</td>
<td>To export these basic messages, select Export Single Messages. Repeat steps 3 and 4.</td>
</tr>
<tr>
<td>7</td>
<td>Now open, e.g. in EXCEL, the message file of the source project and the message file of the destination project. The columns are separated by commas. Compare the structure of the message blocks and, if necessary, make adjustments by reorganizing or renaming columns. Enter the index 0 in each of the blocks with text IDs. This means that the texts are automatically organized in the text library when imported. Under no circumstances may the old ID numbers be retained! The modified file must once again be saved as a text file.</td>
</tr>
<tr>
<td>8</td>
<td>Initiate the import procedure via Messages Import Single Messages.</td>
</tr>
<tr>
<td>9</td>
<td>Now specify the source text file with the exported message information. You now have to decide whether existing messages are to be overwritten during importing. The messages are assigned by means of the respective message number, which must be uniquely defined in the project.</td>
</tr>
<tr>
<td>10</td>
<td>The messages are then imported and supplement your existing message system (Alarm Logging) with the message information already configured. Check the assignments imported.</td>
</tr>
</tbody>
</table>
Note:
If message data is transferred from a WinCC V 1.10 project, you must pay attention to the column headers in the message text file!

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

The message information is already available in an EXCEL table. These messages can be transferred by incorporating the columns into the message structure of the WinCC project. This must be done by building on a WinCC message file. This file is generated as follows:

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

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

3.3.9.9 Transfer of Print Layouts

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

3.3.9.10 Transfer of Global Actions

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

3.3.9.11 Transfer of Project Functions

Copy the desired project functions *.fct from the source folder to the \Library folder of the new project. These functions are made known to the project via the Options -> Regenerate Header menus in the Global Script editor. A detailed description about this topic can be found in the chapter Development Environment for C-Scripts.

3.3.9.12 Application of Standard Functions

In contrast to Project Functions, Standard Functions do not have to be copied. These functions are immediately available to the project, since they are known to all WinCC projects on the station.

3.3.9.13 Transfer of the User Administrator

Since the specification of user groups, users and access rights together with their properties are integrated directly into the database structure, it is not possible to transfer the User Administrator. This means that a reconfiguration is required. There is only one way around this: an automatic data transfer is initiated at the configuration start by through copying an entire basic project.
3.3.10 Operation without a Mouse

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

3.3.10.1 Operation via Keyboard

- A keyboard offers the following input options:
  - Function keys F1 to F12
  - Special function keys (e.g. Operator panel function keys SF10)
  - Standard keyboard inputs
  - Movement to I/O fields or control buttons using the cursor keys or special keys

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

- Control buttons in the plant picture (e.g. for changing pictures)
- by means of function keys
- by means of special keys
- by means of standard keys
- Any key control
- Moving around by means of control objects
- Input fields in the plant picture
- Input/output fields
- special input objects (check-box, ...)
- Alarm Logging (message windows)
- control actions by means of function keys
- control actions by means of specially configured keys
- Tag Logging (trend or table windows)
- control actions by means of function keys
- control actions by means of specially configured keys
- Starting a print job by means of a key
- Logging on or off by means of the keyboard

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

Control actions by means of function keys

Function keys F1 through F12 on the standard keyboard are frequently used as (additional) keyboard action for control buttons for changing pictures in the plant picture hierarchy. These function keys can be assigned at any time to the configured Windows buttons as hotkeys. A hotkey offers you the quickest way of initiating the function assigned to it.
Hotkeys can be assigned, for example, to the functions keys mentioned. These keys are already displayed in the configuration dialog box as selection buttons.

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

The key combination you have selected is displayed in the input field.

This key selection is stored in the properties of the object and can be modified either via the configuration dialog box or directly via the property Miscellaneous → Hotkey.
Action for Hotkey

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

Special Function Keys

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

Standard Keys

If the action is not assigned to a function key input but to a key on the standard keyboard, e.g. the letter m, this key is stored as a hotkey at the Windows Button object.
Any Key Control

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

- Press
- Release

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

Preconfigured button controls (e.g. scrolling through the picture hierarchy) can be found in the optional packages (e.g. Basic Process Control - Picture Tree Manager, etc.). If one of these objects is used as a control element, the button which triggers the action is configured to respond to either the Keyboard - Press or Keyboard - Release event. Either a Direct Connection or a C-Action can be configured as the action. The triggering button event is either

- an any key action or
- a selected key on the standard keyboard

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

3.3.10.2 Movement over Control Objects (Input Fields and Control Fields)

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

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

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

Input/output objects are selected via the alpha cursor (the Tab key or SHIFT+Tab key combination).

All control elements (whether controllable by mouse, keyboard or both) can be integrated into control by means of the tab order. The I/O fields can be integrated into control by means of both the alpha cursor and the tab order.

TAB Sequence

The TAB Sequence (can be set via the Edit → TAB Sequence → Alpha Cursor or → Tab Order) enables you to influence the sequence in which controllable objects are jumped to in runtime mode. The currently selected object can be visualized in runtime. This is the runtime cursor, which can also be turned off (Computer Properties → Graphics Runtime). Buttons configured in the Windows Style are always displayed with a dashed rectangle within the button when selected.

The movement above the controllable elements depends on the settings of the Graphics runtime (Computer Properties → Graphics Runtime).

<table>
<thead>
<tr>
<th>Movement</th>
<th>Standard Keys</th>
<th>Key Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up, Down Left, Right</td>
<td>Cursor keys or Tab (next) or SHIFT+Tab</td>
<td>Other key settings via Computer Properties → Graphics Runtime → Cursor Control Keys</td>
</tr>
<tr>
<td></td>
<td>(previous)</td>
<td></td>
</tr>
<tr>
<td>Alpha Cursor/Tab Order</td>
<td>Tab Order</td>
<td>Switching between alpha cursor and tab order using hotkeys (Computer Properties → Graphics Runtime → Hotkeys) or user-defined keys (using the internal function SetCursorMode)</td>
</tr>
<tr>
<td>Movement in Tables</td>
<td>Normal, i.e. line-by-line editing</td>
<td>This behavior can be changed via Computer Properties → Graphics Runtime → Cursor Control Characteristics</td>
</tr>
<tr>
<td>(Cursor Group)</td>
<td>If the cursor reaches the end of the cursor group, it will remain at this position.</td>
<td></td>
</tr>
</tbody>
</table>

Input/Output Fields

Configured input/output fields can be written in directly after being selected via the keyboard, i.e. you can enter your new data immediately. A pure output field (Properties → Output/Input → Field Type → Output) cannot be operated.

The input is acknowledged - depending on the configured property (Properties → Output/Input → Apply on Exit) - via the ENTER key. In contrast, the ESCAPE key (ESC) exits input without saving the changes (if any) made.
Other Input Objects

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

- **Check-Box**
- **Radio-Button**

The individual selection fields of the check-box or radio-button are set via the spacebar and by moving over the individual components in the box using the Up/Down keys (e.g. arrow keys). This is the key assignment already set by default.

Another input object is the text list object. You can make a selection via a list opened that is dependent on the entries configured:

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

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

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

### 3.3.10.3 Alarm Logging Function Keys for the Toolbar Buttons

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

The most frequent control actions in a message window are

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

When opening the message window or through a further control action, the control must lie in the message window and not in the main window. Depending on the current controllability, the button controls (or function keys) effect the function key bar of the main window or the stored button controls of the message window. This can be achieved, for example, by setting the current focus of control in this section of the window. The focus of control is normally set by clicking with the mouse.

Using the keyboard, the focus can be set in message windows by means of the following configurable routes:

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

Changing (switching) to the message window by means of a quick control action (hot key) which can be used in the same way for all window changes, i.e. in trend windows too, is defined in the startup parameters of the Graphics Runtime. The key combination (e.g. CTRL+W) is entered under Computer Properties ➔ Graphics Runtime ➔ Hotkeys ➔ Window On Top.
Once the message window has been opened, the buttons in the toolbar can be activated directly by means of this key sequence. Direct setting of the focus of control in the message window is, however, achieved by means of the internal function Set_Focus. A C-Action for a button control or open picture command (Picture Object → Events → Miscellaneous → Open Picture) can therefore influence the activation of the message window control. The functions pertaining to the picture focus are available from Internal Functions → graphics → set → focus. For example, for setting the control focus, the following function is called:

\[
\text{Set Focus}(\text{lpwzPictureName}, \text{lpwzObjectName})
\]

As the parameters, the name of the main window (picture name) as well as the Alarm Control (object name) must be specified. A message in the message window is selected by choosing the message line. When the message window is opened, the cursor is positioned in the youngest message (the last message in the message picture). Selecting a message or scrolling in the message window depends on the activation of the scroll mechanism. The picture scroll mechanism can be turned on/off either via a button in the toolbar or be set directly in the configuration of the WinCC Alarm Control.

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

<table>
<thead>
<tr>
<th>Movement</th>
<th>Standard Keys</th>
<th>Key Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up, Down in the Message List</td>
<td>Arrow Keys</td>
<td>Individual message lines</td>
</tr>
<tr>
<td>Beginning, End in the Message List</td>
<td>Pos1 Key, End Key</td>
<td>To the beginning or end of the message list</td>
</tr>
<tr>
<td>Scrolling</td>
<td>Scroll keys (Page Up/Page Down)</td>
<td>Several message lines</td>
</tr>
</tbody>
</table>
In addition to being able to activate the buttons on the toolbar (such as acknowledgment of the selected message) by means of standard mouse action, you can also define control actions by means of function keys.

For each button displayed in the toolbar, a corresponding keyboard control can be configured in the properties.

For Example:

These configuration steps enable you to assign a key combination to each and every one of the buttons used for the message window.
3.3.10.4 Alarm Logging - Toolbar Buttons designed specifically for a Plant

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

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

The functions provided for the button control are located at Standard Functions \rightarrow alarm. For each button in the toolbar, the list contains a function which corresponds to it. For example, the Single Acknowledgment button is called by means of the following function:

\[ \text{AXC_OnBtnSingleAck(1pszPictureName,1pszObjectName)} \]

As the parameter, the window title of the Alarm Logging Control must be entered.

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

A number of examples of such buttons can be found in the optional packages for the alarm system (e.g. Basic Process Control - Horn acknowledgment, etc.).

3.3.10.5 Tag Logging Function Keys for the Toolbar Buttons

In the Tag Logging trend or table controls (trend and table windows) used for displaying measured values, various control buttons are configured for the toolbar, which can be operated by mouse.

- The most frequent control actions in a trend window are
- scrolling through the measured values (time axis)
- selecting a time range
- selecting trends
- using the read ruler

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

When opening the trend window, the control must lie in the trend window and not in the main window. Depending on the current controllability, the button controls (or function keys) effect the function key bar of the main window or the stored button controls of the trend or table window. This can be achieved, for example, by setting the current focus of control in this section of the window. The focus of control is normally set by clicking with the mouse.

Using the keyboard, the focus can be set in trend or table window using the following configuration approaches:
- changing the window by means of a hot key
- setting the focus by means of a control button or

setting the focus directly to a defined element in the trend window upon opening the picture

Implementation of these different variants can be found in this chapter under the description of Alarm Logging.
In addition to being able to activate the buttons on the toolbar (such as selection of a time frame) by means of standard mouse action, you can also define control actions by means of function keys. By default, the individual buttons are occupied by the function keys F1 through F10.

For each button displayed in the toolbar, a corresponding keyboard control can be configured in the properties. For example:

![Properties of WinCC Online Trend Control dialog box]

These configuration steps enable you to assign a key combination to each and every one of the buttons used for the trend or table window. A button control for the trend or table windows must therefore be defined in addition.

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

<table>
<thead>
<tr>
<th>Movement</th>
<th>Standard Keys</th>
<th>Key Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Ruler</td>
<td>Arrow Keys</td>
<td>For moving the read line left or right</td>
</tr>
<tr>
<td>Zooming</td>
<td>For selecting section to be zoomed</td>
<td>For setting and activating substitute input aid for mouse (see system settings)</td>
</tr>
<tr>
<td></td>
<td>INS and arrow keys enable you to define the zoomed window</td>
<td></td>
</tr>
<tr>
<td>Dialog Boxes, e.g. Archive Tag Selection</td>
<td>Tab key</td>
<td>For moving between the input fields</td>
</tr>
<tr>
<td></td>
<td>Arrow Keys</td>
<td>For moving within the tag or tab selection</td>
</tr>
<tr>
<td></td>
<td>+ Key (- Key)</td>
<td>For displaying or closing the tree of the archive tags</td>
</tr>
<tr>
<td></td>
<td>Spacebar</td>
<td>Selection or deselection</td>
</tr>
<tr>
<td></td>
<td>ENTER Key</td>
<td>For confirming and exiting a dialog box</td>
</tr>
<tr>
<td></td>
<td>ESC Key</td>
<td>For canceling a dialog box</td>
</tr>
</tbody>
</table>

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

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

The functions provided for the button control are located at `Standard Functions` → `taglog` → `toolbarbuttons`. For each button in the toolbar, the list contains a function which corresponds to it. For example, the read ruler is called by means of the following function:

```c
TlgTrendWindowPressLinealButton(1pszWindowName);
```

As the parameter, the window title of the Tag Logging Control must be entered. These actions can also be used for buttons you have designed yourself by means of mouse control.
3.3.10.6 Starting a Print Job

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

A printout of the screen contents - a "hardcopy" - can be activated in every picture by means of a hotkey. This hotkey is set globally in the project properties. To do this, select - from the WinCC Explorer - Project Properties → Hotkey → Hardcopy and define the hotkey by directly entering the key combination (by pressing the respective key(s) on the keyboard).

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

ReportJob(pszJobName, pszMethodName);

This function receives the name of the print job as the first parameter. As the second parameter, "PRINT" is transferred, if the print job is to be started immediately, or "PREVIEW", if the print preview is to be displayed.
3.3.10.7 Logging On or Off

In addition to the configurable hotkeys for the log on or log off procedure, a key can also be configured that will display the log on dialog box. You can also log off via a key action. For this purpose, you have to configure a separate button, which can, for example, be activated both by a mouse action and via the keyboard. You can also set a function key control action by means of the hotkey property of the button. The various variations of button controls are described in detail at the beginning of this chapter. The function that is used for logging on or off is a WinCC application function. This function must be configured as a C-Action. Store the C-Action, for example, at the Mouse Action or Press Button event.

The following function is used for logging off:

```c
#include "PWRT_API.H"

PWRTLogout();
```

Logging on is carried out by the function `PWRTLogin()`. Here is an example of how this function could be used:

```c
#include "PWRT_API.H"

PWRTLogin(1);
```

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

<table>
<thead>
<tr>
<th>Movement</th>
<th>Standard Keys</th>
<th>Key Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual input fields</td>
<td>Tab key (forward) or SHIFT+Tab (backward) Arrow Keys</td>
<td>For moving the read line left or right</td>
</tr>
<tr>
<td>Confirm (OK)</td>
<td>ENTER Key</td>
<td>For exiting a dialog box and confirming the input</td>
</tr>
<tr>
<td>Cancel (Abort)</td>
<td>ESC Key</td>
<td>Canceling dialog box or entry</td>
</tr>
</tbody>
</table>
3.3.11 Picture Module Technology

Picture module technology is a crucial strategy for enabling quick and simple configuration and the reusability and maintainability of configured picture components. A configured process box is used, for example, for several process components of the same kind (e.g. valves or controllers). The original picture window configured can now be reused for the control modules that are to be worked with and visualized in the project. This is done in accordance with the following principles:

- copying a picture window and reconnecting the tag fields
- using a picture window whose tag fields are assigned on being called (indirect link)
- application of customized objects with prototypes and resulting objects
- creating prototype pictures and integrating them
- creating OCX picture modules and integrating them as WinCC OCX objects
## Comparing the different Techniques

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
</table>
| Copy of picture windows       | Simple procedure                                                          | All object links must be changed  
Changes to picture buildup lead to complex post-processing |
| Picture windows with indirect connection | • Only one-time configuration of the picture window with simple C-Actions  
• Reuse without copying the basic picture window | Changes to picture buildup lead to complex post-processing |
| Customized objects            | Only one-time configuration of the object with connection using existing Dynamic Wizards | • Changes to picture buildup lead to post-processing, i.e. picture regeneration  
• Cannot be changed centrally |
| Prototype pictures           | • Only one-time configuration of the object  
• Can be changed centrally | (Good) knowledge of C mandatory |
| OCX                           | • Simple integration into configuration of WinCC as an object in the picture  
• A later modification of the OCX object does not lead to post-processing in the objects generated, except when changes are made to the object properties  
• High level of performance  
• Other graphical possibilities  
• Buying-in of new objects (e.g. PCS7 modules) | Must be created by writing a program (C++, VB 5); cannot be created by means of WinCC configuration. |
If only a few simple picture modules are used in the project, one of the first variants is adequate enough to satisfy the person configuring the system. These variants can be implemented without any great training being required. The user object is particularly suitable for simple objects of low to medium complexity and tag link. If you foresee that the object will require a number of changes, you will find it makes sense to get to grips with the concept of prototype pictures. If the graphical blocks are complex or a more comprehensive processing performance is needed, the OCX technology should be preferred. The OCX objects available will in the future grow stronger and stronger in this field. In the following chapters, we will show you the various types of picture module configuration and how they are put to use in the plant pictures. This will enable you to form your own picture of the different variants and their applications in your projects.

3.3.11.1 Process Box as Picture Module

To display the current status of an object (controller, valves, motors, etc.) or to assign set point values, specific information boxes are displayed in the plant pictures. These process boxes typically contain both current states (actual values) and set values, which can be entered by the privileged operator.

Creating the Information Box

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data structures</td>
<td>Define the data structures to be used in the picture module by means of the tag management, e.g. motor with actual value, set value, on-off switch.</td>
</tr>
<tr>
<td>2</td>
<td>Picture module</td>
<td>Using the Graphics Designer, configure a picture which displays the device states, e.g. bars and I/O fields, and control buttons. The size of the picture window (picture object property - X variable and Y variable) must correspond to the target size of the picture window.</td>
</tr>
<tr>
<td>3</td>
<td>Defining the tags</td>
<td>Define the (process) tags in Tag Management, e.g. Motor_T01 of the (structure) data type Motor Type, which is used for the process box.</td>
</tr>
<tr>
<td>4</td>
<td>Tag connection</td>
<td>Now dynamize the individual picture components, e.g. I/O fields, bars etc., by connecting them to the corresponding (process) tags.</td>
</tr>
<tr>
<td>5</td>
<td>Picture window</td>
<td>In the plant picture, create a picture window object and connect it to the picture window contents created under steps 2 through 4 by means of the Picture window name property.</td>
</tr>
<tr>
<td>6</td>
<td>Properties - Settings</td>
<td>This picture window object should not be displayed at the initial opening of the picture. Therefore, the display property must be set statically to no. The appearance of the picture window, with the Windows buttons and title etc., must also be defined in the properties of the picture window.</td>
</tr>
<tr>
<td>7</td>
<td>Calling the picture window</td>
<td>This picture window must be caused to pop up by, e.g. clicking a button or working on the device itself. Design a button which is connected to popping-up of the picture window object (e.g. via a direct connection).</td>
</tr>
</tbody>
</table>
This picture window object, the picture window contents and the corresponding call of the picture window (button) can be reused in a similar form for further devices. All you have to do is copy the picture window object, the picture module and the button. The references must be adapted each time. The picture window object and the button can both be copied by being dragged to and dropped in the graphics library (e.g. project library).

**Customizing the Picture Modules**

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Process tags</td>
<td>Define a new process tag, e.g. Motor_T02, for the defined data structure.</td>
</tr>
<tr>
<td>2</td>
<td>Copy of the picture module</td>
<td>Make a copy of the picture window contents (Motort02.PDL) and change all permanently stored references (e.g. instead of Motor_T01.ActValue, now Motor_T02.ActValue).</td>
</tr>
<tr>
<td>3</td>
<td>Copy of the picture window</td>
<td>Make a copy of the picture window object in the destination plant picture (by dragging and dropping from the graphics library). Adapt the reference to the picture window contents at Properties → Picture Name (Motor02.PDL).</td>
</tr>
<tr>
<td>4</td>
<td>Copy of the button</td>
<td>Make a copy of the button in the destination plant picture (by dragging and dropping from the graphics library). Adapt the reference to the new picture window object in the Direct Connection (Object → Picture Window2 → Display).</td>
</tr>
</tbody>
</table>

In this way, the individual picture windows and their contents can be created for each device and reused by being copied. As can already been seen, the work this necessitates is that of adjusting the permanently stored references for the picture window contents. For this reason, there is a more simple method of achieving reusability through indirect addressing. The amount of adaptation work required should be kept to a minimum.

An alternative solution to this is to configure the picture module without a connection to the picture window. This means that the picture module itself is configured as a non-displayed object in the plant picture. The great disadvantage of this, however, when changing the picture module is that a change must be made in all the pictures in which this picture module is used.
### 3.3.11.2 Picture Module with Indirect Addressing

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

The adaptation and reusability characteristics of a picture module can in this way be simplified considerably.

Configuration is carried out in a similar way to that explained in the steps described above. Here are the actually steps to be carried out:

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specification of data</td>
<td>Definition of the data to be used in the picture module using Tag Management on the one hand (e.g. Motor001_ActValue, Motor001_SetValue, Motor001_Switch), specification of the name container for the individual components to be used in the picture module on the other hand (e.g. ActV_Name, SetV_Name etc.). You initialize these tags with a name, e.g. Motor001_SetValue.</td>
</tr>
<tr>
<td>2</td>
<td>Picture module</td>
<td>Using the Graphics Designer, configure a picture which displays the device states, e.g. bars and I/O fields, and control buttons. The size of the picture window (picture object property - X variable and Y variable) must correspond to the target size of the picture window.</td>
</tr>
<tr>
<td>3</td>
<td>Tag connection</td>
<td>Now dynamize the individual picture components, e.g. I/O fields, bars etc., by connecting them to the corresponding container tags which contain the names of the corresponding tags. You must, however, state in the connection that the tag is only the name of the actual (process) tags. You do this by checking the Indirect Addr. column.</td>
</tr>
<tr>
<td>4</td>
<td>Picture window</td>
<td>In the plant picture, create a picture window object and connect it to the picture window contents created under steps 2 and 3 by means of the Picture window name property.</td>
</tr>
<tr>
<td>5</td>
<td>Properties - Settings</td>
<td>This picture window object should not be displayed at the initial opening of the picture. Therefore, the display property must be set statically to no. The appearance of the picture window, with the Windows buttons and title etc., must also be defined in the properties of the picture window.</td>
</tr>
<tr>
<td>6</td>
<td>Calling the picture window</td>
<td>This picture window must be caused to pop up by, e.g. clicking a button or working on the device itself. Design a button which is connected to popping-up of the picture window object (e.g. via a direct connection).</td>
</tr>
<tr>
<td>7</td>
<td>Graphics library</td>
<td>The picture window object and the button are copied to the library (by being dragged and dropped), so they can be reused.</td>
</tr>
</tbody>
</table>
3.3.11.3 Customized Objects

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data structures</td>
<td>Define the data structures to be used in the picture module by means of the tag management.</td>
</tr>
<tr>
<td>2</td>
<td>Picture module</td>
<td>Using the Graphics Designer, configure a customized object with the user-defined properties.</td>
</tr>
</tbody>
</table>

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

In this dialog box, all the properties of the objects are now declared to be the properties of the customized object, which are later to be dynamized. The basic properties for an object (e.g. the position and size) have already been stored for the customized object. Each individual property of the grouped objects can be selected in the dialog box and be added to the new customized object per drag and drop as a user-defined property or event. Each of these properties can be assigned a new (language-independent) attribute name by the user and also the language-dependent property name (e.g. for configuration in English). Properties which should not be visibly displayed in the properties dialog, but which are used for example in scripts, can be hidden by using the @ character. This means that only a few properties and events (to be dynamized) can be displayed. All the others are hidden.

The customized object you have designed must now be dynamized. To do this, a Wizard is provided:

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Dynamization</td>
<td>Call the dynamic wizard Add dynamics to the prototype. As a template (i.e. prototype), link each individual property of the object to the corresponding structural component of the data structure defined. The structure member for the connection is selected using the tag browser. The wizard, however, only saves the name of the structural component on the property linked (e.g. .Value). Each individual property must be linked separately. This object is now a dynamic object, which, however, has only been linked as a prototype and is not active during runtime. This means that it cannot be updated during runtime.</td>
</tr>
<tr>
<td>4</td>
<td>Copying to the graphics library</td>
<td>Copy this prototype to the graphics library as an object.</td>
</tr>
</tbody>
</table>
The prototype customized object is copied to the graphics library so that it can, for example, be reused again and again. One example of a dynamic object are the pointer instruments in the WinCC library (user library, customized objects, pointer instruments). The prototype object is inserted into the destination plant picture as a copy of the prototype. This copy now has to be linked to the real (process) tags from the tag management.

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Tag</td>
<td>Define a (process) tag for the data structure defined in step 1; this tag is to be used for the customized object.</td>
</tr>
<tr>
<td>6</td>
<td>Creating an instance</td>
<td>Copy the prototype object from the graphics library to the plant picture by dragging and dropping it. Connect this object to the (process) tag via the Dynamic Wizard. The wizard automatically links all necessary structural components of the tags to the correct property of the prototype picture module by replacing each prototype tag link on each property with the actual tag link. You now have and object which is updated with the current tag values during runtime.</td>
</tr>
</tbody>
</table>

### 3.3.11.4 Dynamic Instance

In addition to the Dynamic Wizard Link a prototype to a structure, there is also a Wizard named Make a prototype dynamic. How does it differ from linking to a structure and which steps have to be modified?

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Picture module</td>
<td>Using the Graphics Designer, configure a customized object with the user-defined properties as described above. The customized object must contain a Static Text component, whose Text property is transferred as a user-defined component. This text property is assigned the attribute name <code>TagName</code>. This tag name is used for dynamic linking of the (process) tags.</td>
</tr>
<tr>
<td>2</td>
<td>Tag</td>
<td>Define a (process) tag for the data structure defined in step 1; this tag is to be used for the customized object.</td>
</tr>
</tbody>
</table>
3 Creating an instance

Copy the prototype object from the graphics library to the plant picture by dragging and dropping it. Connect this object to the (process) tag via the Dynamic Wizard link a prototype to a structure: The wizard automatically links all necessary structural components of the tags to the correct property of the prototype picture module by replacing each prototype tag connection on each property with the actual tag connection. You now have an object which is updated with the current tag values during runtime.

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

### 3.3.11.5 Prototype Pictures

The technology of the prototype pictures goes one step further. When prototypes are used, the concept can be structured so flexibly that a change made to the prototype is automatically followed by an adjustment to the objects created. This flexibility requires the linking of very flexible C scripts.

The technology of the prototype pictures operates using template pictures which can be repeatedly integrated into one or more parent pictures. A template picture is only a template, which is "brought to life" only once integrated into a real object. An object based on a template (= prototype picture) is brought into being by what it known as an "instantiation". A number of instances (i.e. real objects) can be created for one template.

**Picture Window Template**

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

In a parent picture, up to 30 instances (i.e. objects) of a particular template type can be displayed. If different prototypes are used, it is possible to use more than 30 objects. The prototype pictures are picture modules which are copied to the library after being created, so they can be reused. The picture modules used are used in the plant pictures as instances of the template. These copies show the current data, e.g. of controllers or motors, which is visualized in the module. The corresponding controller or motor components are displayed automatically.

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data structures</td>
<td>Define the data structures to be used in the picture module by means of the tag management.</td>
</tr>
<tr>
<td>2</td>
<td>Picture window buildup</td>
<td>In the Graphics Designer, configure the content of the picture module, e.g. bars, I/O fields, etc.</td>
</tr>
<tr>
<td>3</td>
<td>Linking data structure with picture window buildup</td>
<td>Link the picture components with the individual data structure components with special C scripts (from the sample project).</td>
</tr>
<tr>
<td>4</td>
<td>Picture window</td>
<td>Link the picture module to the picture window object.</td>
</tr>
<tr>
<td>5</td>
<td>Graphics library</td>
<td>Copy the picture window to the graphics library.</td>
</tr>
</tbody>
</table>

The picture module can now be used in the plant pictures by being retrieved from the graphics library.

Let's first take a look at how the template picture is created.

<table>
<thead>
<tr>
<th>Step</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creating a tag structure (structure data type) in the data manager; this is where the number of tags that make up the structure is defined (member tags), the names of the tags and their data type (BIT, SHORT, etc.). For example, PID with the structure components set point value, actual value and temperature.</td>
</tr>
<tr>
<td>2</td>
<td>Creating a picture which is to be used as a picture module. This is usually smaller than the size of the monitor screen and can be dimensioned accordingly. Each picture that has been created in its own right can be used to create a template. The picture is created using the graphics editing functions, and graphics tags such as I/O fields, bars etc. are positioned in it but not linked to tags. Internal relationships (direct links) between graphic objects, such as change-controlled transfer of the output value of an I/O field to a bar, are configured in this picture.</td>
</tr>
</tbody>
</table>
### Configuration

<table>
<thead>
<tr>
<th>Step</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>The graphic fields are now linked to the structural components of the corresponding tag structure. This link configuration links the tags at type level (template only), but not yet with concrete process objects. For this purpose, you will find a pre-prepared sample project on the WinCC CD (\Samples). In the project library (\Template) of this project is a customized object called TemplateInit, which performs this connection. This is located in the graphics library and can be dragged and dropped from here into a picture that is to be standardized. TemplateInit already has a complete script logic. This logic uses what is known as a ConnectionTable, which during the configuration is filled in as a table and contains precisely the underscored entries quoted above. This is method used to define the link between the properties and the structural components. The buildup of these links can be set within a template or also from outside. To this end, the special project graphics library contains customized objects which to the naked eye look like simple buttons, but which in fact contain parameterizable information about the template to be called up. All of the scripts for implementing this already prepared generation of the prototype must be copied to the destination project. See the final steps described at the end of the chapter under steps 8 - 10. Without these project functions, these prototypes cannot be implemented.</td>
</tr>
<tr>
<td>4</td>
<td>This picture is to be used as a process box. To do this, create a picture window object in a temporary picture (i.e. it is only required for this intermediate step) and connect the Picture Name property of this object to the picture that contains the picture module.</td>
</tr>
<tr>
<td>5</td>
<td>Copy this picture window object to the graphics library by dragging and dropping it.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Defining the tags</td>
<td>Define a (process) tag for the corresponding data type (e.g. PID_1 of type PID).</td>
</tr>
<tr>
<td>7</td>
<td>Creating an instance</td>
<td>Copy the picture window module from the graphics library. Assign the picture Window - object name the name of the (process) tag that is being used (e.g. PID_1): Set Picture Window Object ➔ Picture Window ➔ Object Name ➔ Static to PID_1</td>
</tr>
</tbody>
</table>

hen the picture module is positioned in the picture, it is given the name of a structured (process) tag whose values contain the status data of a process object. During runtime, then, the picture module retrieves the status data automatically. A number of C-Scripts are used for these prototype pictures, which are or have been stored as project functions. To be able to use the C scripts that have already been prepared, you have to adopt the following scripts from the sample project. Proceed as follows:
<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Copying the function files</td>
<td>From the path</td>
</tr>
<tr>
<td>9</td>
<td>Making known in the project</td>
<td>In WinCC, start the Global Script editor (Global Script (\text{Open})) to make these new functions known in the following manner: you can now make the new functions known to the function tree of the project functions by clicking on the Regenerate Header button. The new functions can now be seen in the list of project functions.</td>
</tr>
<tr>
<td>10</td>
<td>Transferring customized objects</td>
<td>The customized objects are made available by means of the project library. In a new project in which you have so far not stored any of your own symbols in the library, you can simply copy the library from the sample project to your project: Copy to your path (\text{Library!} ) Otherwise, use the export mechanism to transfer the customized objects from one project to the other. Export the desired symbols in the sample project as .emf files (File (\text{Export})) and import these .emf symbols in your own project into a temporary picture via Insert (\text{Import}). Transfer the symbols to your project library by dragging and dropping them. Use a separate folder also to do this, e.g. Template.</td>
</tr>
</tbody>
</table>

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

- When using these prototype pictures, there are different ways in which the picture module components are supplied with the current names. This is done by means of the following variations, which we will briefly name here:
  - The instance name is determined upon the selection of the picture window: for this, a predefined project function (EnableTemplateInstance) is stored at the open picture event in the picture window.
  - The instance name is defined via an input/output tag, which is automatically read in the picture window via a script that determines the instance name: this is done by using the prepared customized object \(\text{InstanceCallButton+Template}\).
  - A button transfers the instance name directly to the called picture window: this is done by using the prepared customized object \(\text{InstanceCallButtons+Template}\).

You can find detailed examples of these variations in the sample project on the WinCC CD-ROM (\$Samples).
### 3.3.11.6 OCX Objects

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inserting an OCX object</td>
<td>From the Object Palette of the Graphics Designer at smart objects, select the entry OLE Control (OCX). Drag the object into the plant picture by holding down the , from the displayed dialog box, select the desired element.</td>
</tr>
<tr>
<td>2</td>
<td>Connecting properties</td>
<td>The object inserted likewise has properties and events. Which properties and events are available depend on the specific OCX. A connection to a tag can be created, e.g. at the property process driver connection.</td>
</tr>
</tbody>
</table>

### Creation

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

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

### Registration

The OCX modules created or purchased in must be registered on the respective WinCC station. You can see which OCX objects are available on the WinCC station by looking in the selection dialog of the Graphics Designer (see description above). All the OCX elements registered on the computer are listed in the dialog box. An OCX element is stored on the computer in the form of a file with the extension .OCX or .dll.

If a module has not yet been registered, you can do so in the WinCC OLE Control dialog box. The dialog box contains a button for registering as well as removing the registration of the currently selected component.

The relevant file must be on the WinCC station for registration to be carried out. The compatibility and functionality of the OCX components must be tested by the configurator himself. Only those OCX modules marked using WinCC have been used and tested in the WinCC environment.
3.3.12 Online Configuration (Runtime) - Notes, Limitations

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

WinCC Explorer

- The following changes are not adopted:
  - changing the type of a computer in the computer list
During runtime, the following configuration steps are not possible:
  - deleting/renaming tags
  - changing the data type of a tag

Alarm Logging

The following changes are not adopted:

- changing the archives/reports
- changing the group messages
- every message after a total of 500 single messages while runtime is active
During runtime, the following configuration steps are not possible:

  - no limitations

Tag Logging

The following changes are not adopted:

  - no limitations.
  - During runtime, the following configuration steps are not possible:
    - tables of user archives can be created but not changed
    - deleting data in Tag Logging and from user archives
Exceptions for the configuration during runtime:
the runtime API of Tag Logging can be used to edit and delete the tables of the user archives

Global Script

The following changes are not adopted:

- changes made to a Wizard script are only adopted after restarting the Graphics Designer modified Wizard scripts
During runtime, the following configuration steps are not possible:

  - no limitations
Report Designer
The following changes are not adopted:
• changes made to the message sequence report, since once started, it always remains active in runtime and does not reload the layout information
During runtime, the following configuration steps are not possible:
• no limitations

Redundancy
The following changes are not adopted:
• the computer name of the partner cannot be transferred to a third computer
• the AutoSwitcher cannot be changed, i.e. you must configure at the beginning to where the AutoSwitcher is to switch. It does, however, also switch back if the other one fails
During runtime, the following configuration steps are not possible:
• no limitations

SIMATIC S7 Protocol Suite or S/7PMC Channel
The following changes are not adopted:
• no diagnostics parameters relating to the S7Chn.ini (not released) are adopted online although all changes to communication addresses are adopted online, they are only evaluated when a connection is set up
During runtime, the following configuration steps are not possible:
• no limitations

Text Library
The following changes are not adopted:
• no limitations.
• In the Text Library, the changed texts are adopted via File → Send Changes to Active Project
• In Alarm Logging, the changes are copied to the Text Library via File → Save
During runtime, the following configuration steps are not possible:
• no limitations

User Administrator
The following changes are not adopted:
changes to the user authorizations only become effective after logging off/on again
• During runtime, the following configuration steps are not possible:
• no limitations
4 WinCC C-Course

To make objects dynamic, various options are available in WinCC. Among others, these are tag connections, dynamic dialogs and direct connections. With these aids, the implementation of complex dynamics is possible. Nevertheless, they reach their limits with growing demands. A much broader range of possibilities is available to users through the configuration of C-Actions, project functions or actions. They are created in the WinCC script language C. For many applications it is not necessary to have a very comprehensive knowledge of C. It is sufficient to supply existing functions with parameters. However, to use all capabilities of C as the WinCC script language, a basic knowledge of this programming language is needed. This course can provide you with this knowledge.

Target Group

This course is intended to provide basic knowledge about the general application of the programming language C to everybody who is not familiar with it. Experienced C programmers can learn the special features of C when applied to WinCC. The sample project for this course can be copied directly from the online document to your hard drive. By default, it will be stored to the C:\Configuration_Manual folder.
Sample Project

The interface of the sample project is divided into several sections. They are listed below:

- **Navigation Bar (1):** The navigation bar allows the selection of the pictures pertaining to the various chapters.

- **Chapter Window (2):** The chapter window displays the pictures assigned to the individual chapters. These pictures contain all samples described in the particular chapter. Most of these samples are created at buttons.

- **Script Window (3):** The script window displays the code of the sample currently selected in the chapter window. The currently selected sample is marked red in the chapter window.

- **Diagnostics Window (4):** The diagnostics window displays all outputs of the various samples initiated by the printf() function.
4.1 Development Environment for C-Scripts

For the creation of C-Scripts, WinCC provides two different editors. One is the action editor in the Graphics Designer for the creation of C-Actions at objects, the other is the Global Script editor for the creation of project functions and global actions. The syntax of the script language corresponds to Standard C following ANSI. Another area of application for the programming language C in WinCC pertains to the creation of Dynamic Wizards. For this, a separate editor is available. The usage of this editor is explained in the sample pertaining to the Dynamic Wizard and will not be treated in the general overview.
### 4.1.1 Action Editor of the Graphics Designer

In the Graphics Designer, object properties can be made dynamic via C-Actions. In the same way, C-Actions can be used to react to object events.

#### Action Editor

For the configuration of a C-Action, the action editor is available. This editor is opened from the Object Properties dialog box via a button on the desired property or event and then selecting C-Action from the displayed pop-up menu. Already existing C-Actions are marked by a green arrow at the property or event.

In the action editor, the C-Action can be programmed. For C-Actions at properties, a trigger must be defined. For C-Actions at events, this is not necessary since the event itself is the trigger. The completed C-Action must be compiled. If no errors are detected by the compiler, the action editor can be exited by clicking on OK.
Structure of a C-Action

In general, a C-Action corresponds to a function in C. There are two different types of C-Actions: Actions that are created at Properties and actions that are created at Events.

Generally, a C-Action at a property is used to control the value of this property with respect to different environmental conditions (e.g. the value of a tag). For this type of C-Action, a trigger must be defined which controls its execution. A C-Action at an event is used to react to this event.

C-Action at a Property

```c
#include 'spclpp.h'
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    /*1*/ long lReturnValue;
    /*2*/ lReturnValue = GetTagEDword("SIZZ_course_test_1");
    /*3*/ return lReturnValue;
}
```

The sample code above represents a typical C-Action at a property. The meaning of the individual sections is described below.

- **Header (gray):** The first three lines shaded gray form the header of the C-Action. This header is generated automatically and cannot be changed. Except for the return value type (long in the sample code), the function header is identical for all properties. Three parameters are transferred to the C-Action. These are the Picture Name (lpszPictureName), the Object Name (lpszObjectName) and the Property Name (lpszPropertyName).

- **Variable Declaration (1):** In this first code section that can be edited, the variables used are declared. In the sample code this is one variable of the long type.

- **Value Computation (2):** In this section, the computation of the property value is performed. In the sample code, only the value of one WinCC tag is read in.

- **Value Return (3):** The computed property value is assigned to the property. This is done via the return command.
C-Action at an Event

```c
#include "opdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    /*1*/ long lValue;
    /*2*/ lValue = GetTagSWord("S32i_course_test_1");
    SetInt(lpszPictureName, lpszObjectName, lValue);
}
```

The sample code above represents a typical C-Action at an event. The meaning of the individual sections is described below.

- **Header (gray)**: The first three lines shaded gray form the header of the C-Action. This header is generated automatically and cannot be changed. The function header differs for the various types of events. The parameters `lpszPictureName` (Picture Name), `lpszObjectName` (Object Name) and `lpszPropertyName` (Property Name) are transferred to the C-Action. The `lpszPropertyName` parameter only contains relevant information for events that react to the change of a property. Additional event-specific parameters can be transferred.

- **Variable Declaration (1)**: In this first code section that can be edited, the variables used are declared. In the sample code this is one variable of the `long` type.

- **Event Processing (2)**: In this section, the actions reacting to the corresponding event are performed. In the sample code, the value of one WinCC tag is read in. This value is assigned as position X to the own object. The return value of a C-Action at an event is of the `void` type, i.e. no return value is expected.
Creation of a C-Action

The following table describes the individual steps required to create a C-Action.

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a C-Action</th>
</tr>
</thead>
</table>
| 1    | Open the Graphics Designer editor.  
      | Open the desired WinCC picture.   
      | Open the Object Properties dialog box of the desired object. |
| 2    | The *Edit Action* editor is opened by R on the desired property or event and then selecting *C-Action* from the pop-up menu. |
| 3    | The *Edit Action* editor will be displayed.  
      | In it, the basic framework of a function will be displayed.  
      | Among other things, the header of the *C-Action* has been automatically generated. This header cannot be changed.  
      | In the first line of the *C-Action*’s header, the file *apdefap.h* is integrated. Through this file, all project functions, standard functions as well as internal functions are announced to the *C-Action*.  
      | The second part of the *C-Action*’s header is the function header. The function header provides information about the return value of the *C-Action* and transferred parameters which can be used within the *C-Action*.  
      | The third part of the *C-Action*’s header is the open bracket. This bracket cannot be deleted. Between this open bracket and the closing bracket, the actual code of the *C-Action* is programmed. |
Step | Procedure: Creation of a C-Action
---|---
4 | Another automatically generated code section consists of two comment blocks. These blocks are necessary to give the CrossReference editor access to the internal information of a C-Action. They are also needed to enable the rewiring within a C-Action. If both of these options are not going to be utilized, these comments can also be deleted. The first comment block is used for the definition of the WinCC tags used in the C-Action. In the program code, the defined name of the variables instead of the real variable names must be used. The second comment block is used for the definition of the WinCC pictures used in the C-Action. Here as well the defined name of the picture must be used in the program code instead of the real picture name. A sample code pertaining to this topic can be found following this table. It includes the definitions of a WinCC tag and a WinCC picture as well as the subsequent application of these definitions.

5 | Programming the function body, which executes the desired calculations, actions and so forth. Several programming aids are available. One of these aids is the tag selection dialog. This dialog is opened via the toolbar button displayed below. In the displayed Select Tag dialog, a WinCC tag is selected and then confirmed by clicking on OK. The name of the selected WinCC tag will then be inserted at the current cursor position in the C-Action.

6 | Another aid is the function selection in the left window of the action editor. Using the function selection, all available project functions, standard functions and internal functions can be inserted automatically at the current cursor position in the C-Action.

For this, the desired function is selected via a . This will display the Assigning Parameters dialog, which contains a list of all parameters that must be fed and their data types. The function can be parameterized in the Value column. In addition to the plain text input, the options Select Tag, Graphic Objects and Pictures are available. To insert the function at the current cursor position in the C-Action, confirm the dialog by clicking on OK.

<table>
<thead>
<tr>
<th>Assigning Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Name</td>
<td>SetTagSDWord</td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Data Type</td>
</tr>
<tr>
<td>Tag Name</td>
<td>Tag value</td>
</tr>
<tr>
<td>Description</td>
<td>Data Type</td>
</tr>
<tr>
<td>Graphic objects</td>
<td>Pictures</td>
</tr>
</tbody>
</table>

OK Cancel
The completed function must now be compiled. This is done via the toolbar button displayed below.

The result of the compiling process is displayed in the lower left corner of the action editor. This includes the number of errors found and the number of warnings. Errors always cause a C-Action to be not executable. Warnings on the other hand are notes pointing out possible errors that can occur during the execution of the C-Action. Good programming style precludes the creation of C-Actions with the output result other than 0 Error(s), 0 Warning(s).

If errors occur during the compiling process, they will be displayed in the output window. Via a D on an error message in the output window, you can jump directly to the corresponding code line.

For C-Actions that have been created at an object property, a trigger must be defined. For C-Actions at events, this is not necessary since the event itself forms the trigger.

The definition of the trigger is carried out via the toolbar button displayed below. You have the option to use time or tag triggers.

By clicking on the OK button of the action editor, the programmed C-Action will be placed at the desired property or event. A property or event made dynamic through a C-Action will be marked by a green arrow.
Defining WinCC Tags and WinCC Pictures

```c
#include "opengl.h"
long main(char* lpszObjectName, char* lpszObjectOwner, char* lpszPropertyName) {
    // WINCC:TAGNAME SECTION START
    // syntax: #define TagNameInAction 'TagName'
    #define S32I_COURSE_TEST_1 "S32i_course_test_1"
    // WINCC:TAGNAME SECTION END
    
    // WINCC:PICNAME SECTION START
    // syntax: #define PICNameInAction 'PicName'
    #define CC_0_STARTPICTURE_00 "cc_0_startpicture_00.png"
    // WINCC:PICNAME SECTION END
    
    SetTagSDWord(S32I_COURSE_TEST_1, 100);
    OpenPicture(CC_0_STARTPICTURE_00);
    return 0;
}
```

If a new C-Action is created, the automatically generated code will include two comment blocks. These comment blocks are necessary to give the CrossReference editor access to the internal information of a C-Action. They are also needed to enable the rewiring within a C-Action.

- **Variable Definition**: The first comment block is used for the definition of the WinCC tags used in the C-Action. This comment block begins with the line `// WINCC: TAGNAME SECTION START` and ends with the line `// WINCC: TAGNAME SECTION END`. In between these two lines, the names of all WinCC tags used in the C-Action are defined. A definition takes place through the preprocessor command `#define` followed by the defined name (in the sample code this is `S32I_COURSE_TEST_1`) followed by the name of the WinCC tag (in the sample code this is `S32i_course_test_1`).

- **Picture Definition**: The second comment block is used for the definition of the WinCC pictures used in the C-Action. This comment block begins with the line `// WINCC: PICNAME SECTION START` and ends with the line `// WINCC: PICNAME SECTION END`. In between these two lines, the names of all WinCC pictures used in the C-Action are defined. This follows the same convention as described above for the definition of tag names.

- **Application**: In the actual program code, the defined values must be used instead of the real tag and picture names. Before compiling the C-Action, the preprocessor will replace all defined names with the real names.
4.1.2 The Global Script Editor

The Global Script editor is used for the creation of project functions, standard functions and actions.

Project Functions

If the same functionality is frequently required in C-Actions, this functionality can be formulated in a project function. Project functions can be called in all C-Actions of a WinCC project, in the same manner as all other functions. The following lists the advantages of using project functions as opposed to creating the entire program code in C-Actions:

- **Central place for editing**: A change of a project function affects all C-Actions in which this function is being used. If no project functions are used, all concerned C-Actions must be changed manually. This not only simplifies the configuration, but also maintenance and troubleshooting.

- **Reusability**: Once a project function has been programmed and extensively tested, it can be used again at any time without requiring additional configurations or new tests.

- **Reduced Picture Volume**: If not the entire program code is placed directly in the C-Action at the object, the picture volume is reduced. This results in faster picture opening times and a higher performance in runtime.

- **Password Protection**: Project functions can be protected against changes by assigning a password. This protects the configuration data as well as your know-how.

Project functions are only available project-internal. They are stored in the WinCCProjectFolder\LIBRARY folder and defined in the ap_pbib.h file located in the same folder.

A number of standard functions are available. Contrary to the project functions, the standard functions are available to all WinCC projects. Existing standard functions can be changed. New standard functions can also be created.

Standard functions only differ from project functions with regard to their availability: standard functions are available across projects, whereas project functions are only available project-internal. Standard functions are stored in the WinCCInstallationFolder\APLIB folder and defined in the ap_glob.h file located in the same folder.

Internal Functions

In addition to the project functions and the standard functions, there are also internal functions. Among others, these are standard C functions. They cannot be changed by the user and new internal functions can also not be created.
Actions

Actions are - contrary to the previously described functions - not called from a C-Action or other function. A trigger must be assigned to the action controlling its execution. It is executed independently of the currently selected picture in runtime.

Actions can be configured globally, i.e. across projects. In this case, they are stored in the WinCCProjectFolder\PAS folder. You can also configure local actions (computer-specific actions), which will be stored in the WinCCProjectFolder\ComputerName\PAS folder.

If Global Script Runtime is checked in the startup list of the computer, all global actions and all local actions belonging to the computer will activated upon project start.

Creation of a Project Function

The steps required to create a project function are identical to the steps required to create a standard function. The following description therefore also applies to the creation of a standard function.

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Project Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Global Script editor.</td>
</tr>
<tr>
<td>2</td>
<td>Via a right-click on the Project Functions entry and then selecting New → Function from the pop-up menu, the basic framework of a new project function will be created.</td>
</tr>
<tr>
<td>Step</td>
<td>Procedure: Creation of a Project Function</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>The <em>project function</em> can be completely formulated by the user. There are no code sections that cannot be edited. Programming the function header. The function must be given a name, which is used in <em>C-Actions</em> or other functions to call it. Additionally, the return value and required transfer parameters of the function must be specified. If other <em>project functions</em> or <em>standard functions</em> are to be used in the present function, the <em>apdefap.h</em> file must be integrated. This is done via the preprocessor command <code>#include &quot;apdefap.h&quot;</code>, which must be inserted before the function header.</td>
</tr>
<tr>
<td>4</td>
<td>Programming the function body. For this, the same aids as for programming <em>C-Actions</em> are available. Specifically, these are the tag selection and function selection.</td>
</tr>
<tr>
<td>5</td>
<td>The completed function must now be compiled. This is done via the toolbar button displayed below. The result of the compiling process is displayed in the output window. Errors occurred and warnings will be listed and their quantity displayed. Via a button on an error message in the output window, you can jump directly to the corresponding code line.</td>
</tr>
<tr>
<td>6</td>
<td>Via the toolbar button displayed below, a description can be added to the <em>project function</em>. Together with the description, a password can be defined to protect the <em>project function</em> from unauthorized access.</td>
</tr>
<tr>
<td>7</td>
<td>The completed <em>project function</em> must be saved using an appropriate name.</td>
</tr>
</tbody>
</table>
Creation of a Global Action

The steps required to create a global action are identical to the steps required to create a local action. The following description therefore also applies to the creation of a local action.

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Creation of a Global Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Global Script editor.</td>
</tr>
<tr>
<td>2</td>
<td>Via a on the Global Actions entry and then selecting New Action from the pop-up menu, the basic framework of a new action will be created.</td>
</tr>
<tr>
<td>3</td>
<td>The header of the action will be generated automatically and cannot be changed. In addition, two comment blocks for the definition of the WinCC tags and WinCC pictures are inserted. The meaning of these two comment blocks has already been described in the previous C-Actions section.</td>
</tr>
</tbody>
</table>
Procedure: Creation of a Global Action

4 Programming the action body.

For this, the same aids as for programming C-Actions are available. Specifically, these are the tag selection and function selection.

The action has a return value of the type int. This return value, however, cannot be evaluated by the user. By default, the value 0 is returned.

5 Via the toolbar button displayed below, a description can be added to the action, similar to the function description. A password can also be defined to protect the project function from unauthorized access.

Contrary to functions, it is also necessary to set a trigger, which controls the execution of the action. For the selection of a trigger for an action, the user has a greater range of options than for the selection of a trigger for a C-Action at an object. Among other things, a one-time execution can be programmed.

6 The completed action must be saved.

Test Output

The execution of a program can be followed through test outputs. This facilitates troubleshooting and error diagnostics during development. Test outputs can be initiated via the printf() functions. Through this function, simple texts, but also current tag values, can be output. To make the output texts visible, a Global Script Diagnostics Window must be configured.
The printf() Function

The printf() function allows test outputs to be made. A sample application of this function is displayed below:

```c
printf("I am %d years old\n", age);
```

The printf() function requires at least one parameter. This parameter is a character string. The type and quantity of additional parameters to be transferred depends on this character string.

The % character is used by the printf() function as an identifier that the value of a variable is to be inserted at this position. The character following the % character determines the data type of this variable. The character combination %d used in the sample above signals the output of a decimal number. Additional possible combinations and their descriptions can be found in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>Output of a decimal number (int or char)</td>
</tr>
<tr>
<td>%ld</td>
<td>Output of a variable of the type long as a decimal number</td>
</tr>
<tr>
<td>%c</td>
<td>Output of a character (char)</td>
</tr>
<tr>
<td>%x</td>
<td>Output of a number in hexadecimal format (with lower case a...f)</td>
</tr>
<tr>
<td>%X</td>
<td>Output of a number in hexadecimal format (with upper case A...F)</td>
</tr>
<tr>
<td>%o</td>
<td>Output of a number in octal format</td>
</tr>
<tr>
<td>%u</td>
<td>Output of a decimal number (only for unsigned types)</td>
</tr>
<tr>
<td>%f</td>
<td>Output of a float value in floating point notation, e.g. 3.43234</td>
</tr>
<tr>
<td>%e</td>
<td>Output of a float value in exponential notation, e.g. 23e+432</td>
</tr>
<tr>
<td>%E</td>
<td>Like %e, but with upper case E, e.g. 23E+432</td>
</tr>
<tr>
<td>%s</td>
<td>Output of a character string (char*)</td>
</tr>
<tr>
<td>%le</td>
<td>Output of a double value</td>
</tr>
<tr>
<td>%%</td>
<td>Output of a % character</td>
</tr>
<tr>
<td>\n</td>
<td>Output of a line change (carriage return)</td>
</tr>
<tr>
<td>\r</td>
<td>Output of a line feed</td>
</tr>
<tr>
<td>\t</td>
<td>Output of a tab</td>
</tr>
<tr>
<td>\</td>
<td>Output of a \ character</td>
</tr>
</tbody>
</table>
Global Script Diagnostics Window

The text outputs specified by the printf() function are displayed in the Global Script Diagnostics Window. The steps for the configuration of such a Diagnostics Window are described in the following table:

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Global Script Diagnostics Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Graphics Designer editor.</td>
</tr>
<tr>
<td></td>
<td>Open the desired WinCC picture.</td>
</tr>
<tr>
<td>2</td>
<td>Configure a Smart Object ➔ Application Window.</td>
</tr>
<tr>
<td></td>
<td>After placing the application window in the picture, the Window Contents dialog will be opened. From the list-box, select the Global Script entry. Exit the dialog by clicking on OK. The Template dialog will be opened. From the list-box, select the GSC Diagnostics entry. Exit the dialog by clicking on OK as well.</td>
</tr>
<tr>
<td>3</td>
<td>In order to work comfortably with the Global Script Diagnostics Window, it is recommended to set all properties in the Object Properties dialog under the Miscellaneous entry to Yes.</td>
</tr>
</tbody>
</table>

![Object Properties](image)
<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Global Script Diagnostics Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>If the project is in runtime, the text outputs specified by the <code>printf()</code> function will be displayed in the diagnostics window. If the update is stopped using the appropriate button on the toolbar, the window content can be saved or printed out.</td>
</tr>
</tbody>
</table>

![Global Script - Diagnose](image)

Error in `GetOpenFileName()`
4.2 Variables

In the WinCC project Project_C_Course, samples pertaining to the topic variables can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_00.PDL picture.

Variables

Variables are data objects manipulated by a program. A variable can only be used after it has been defined. All variables used in a program must be defined before the first instruction can be carried out.

Variables can be compared to a container. Through a variable name, we give the container a unique name. The type of the content of the container is specified by its data type. The start content of the container is specified by its initialization value. In most cases, this content will be manipulated during program execution.

The variables described here should not be mistaken for WinCC tags. They are only available within the program code.

A sample for the definition of a variable is illustrated by the following program code. In this case, a variable of the int data type is defined with the name iNumber. The code line is concluded with a semicolon. In front of the variable name is a prefix describing the data type. This is not mandatory, but it makes the data type of the variable immediately recognizable during program creation.

```c
int iNumber;
```

It is also possible to initialize a variable while it is being defined.

```c
int iNumber = 0;
```

Constants

In addition to variables, a program also works with constants. This is simply the direct application of a number value. To clarify the meaning of such a number value, a symbolic constant can be defined for it using the `#define` command.

A sample for the definition of a symbolic constant is shown by the program code below. In this case, the symbolic constant MAX_INT_VALUE is defined with the number value 2147483647. Note that the code line must not be concluded by a semicolon. It is a common programming guideline that symbolic constants are written in upper case letters to easier differentiate them from variables.

```c
#define MAX_INT_VALUE 2147483647
```
Data Types

C recognizes the basic data types listed in the table below.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>One Byte, can accept one character</td>
</tr>
<tr>
<td>int</td>
<td>Integer value</td>
</tr>
<tr>
<td>float</td>
<td>Floating-point number single accuracy</td>
</tr>
<tr>
<td>double</td>
<td>Floating-point number with double accuracy</td>
</tr>
</tbody>
</table>

A variable of the char data type has a memory requirement of one Byte. Its content can be interpreted as a character or as a number. The int data type can be preceded by the signed or unsigned keyword. The keyword signed stands for a signed value, the keyword unsigned for an unsigned value. The int data type can also be preceded by the long or short keywords. These keywords can also be used without int - and still have the same meaning. A variable of the short (or short int) data type has a memory requirement of 2 Bytes, a variable of the long (or long int) data type has - just as the variable of the int data type - a memory requirement of 4 Bytes. The double data type only differs from the float data type by its value range. Numbers can be represented with greater accuracy by the double data type. A variable of the float data type has a memory requirement of 4 Bytes, whereas a variable of the double data type has a memory requirement of 8 Bytes.

Value Ranges of Data Types

Each data type is capable of displaying number values in a certain value range. Differences arise from the different memory requirements of the different data types and the difference between signed and unsigned data types.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-2 147 483 648 to 2 147 483 647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>0 to 4 294 967 295</td>
</tr>
<tr>
<td>short</td>
<td>-32 768 to 32 767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>0 to 65 535</td>
</tr>
<tr>
<td>long</td>
<td>-2 147 483 648 to 2 147 483 647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>0 to 4 294 967 295</td>
</tr>
<tr>
<td>char</td>
<td>-128 to 127 (all ASCII characters)</td>
</tr>
<tr>
<td>unsigned char</td>
<td>0 to 255 (all ASCII characters)</td>
</tr>
<tr>
<td>float</td>
<td>-10 ^38 to 0 ^38</td>
</tr>
<tr>
<td>double</td>
<td>-10 ^308 to 0 ^308</td>
</tr>
</tbody>
</table>
4.2.1 Sample 1 - C Data Types (Integers)

In this sample, the available default data types of C are used to display integers. The sample has been configured at the Button1 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button1

```c
#include "apchap.h"
void OnClick(char* lpstrPictureName, char* lpstrObjectName, char* lpstrProperty)
{
    char cNumber;      //signed 8 bit value
    long lNumber;      //signed 32 bit value
    short sNumber;     //signed 16 bit value
    int iNumber;       //signed 32 bit value

    unsigned char ucNumber; //unsigned 8 bit value
    unsigned long ulNumber; //unsigned 32 bit value
    unsigned short usNumber; //unsigned 16 bit value
    unsigned int uiNumber; //unsigned 32 bit value

    cNumber = -128;
    sNumber = -32768;
    iNumber = -2147483648;
    lNumber = 2147483647;

    //output in diagnostics window
    printf("%d\r\n", cNumber);
    printf("%d\r\n", sNumber);
    printf("%d\r\n", iNumber);
    printf("%d\r\n", lNumber);
    printf("%d\r\n", ucNumber);
    printf("%d\r\n", usNumber);
    printf("%d\r\n", ulNumber);
    printf("%d\r\n", uiNumber);
}
```

- The first three lines are the header of the C-Action. This header cannot be changed.
- In the next section, the variables are defined. One variable each of the `char`, `long`, `short` and `int` data type and their unsigned counterparts are defined. The names of the variables are preceded by a prefix describing the data type. This is not mandatory, but it makes the data type of a variable immediately recognizable during program creation. As the comment, each line includes the memory space required by the variable (comments begin with the character string // are marked in green).
- In the next section, values are assigned to the variables. This is done using the assignment operator =. The number values used in the sample exactly hit the limits of the value ranges that can be displayed for the various data types.
- The number values are output in the diagnostics window through the function `printf()`. This output is displayed in the next section.
Output in the Diagnostics Window

The sample described in this section generates the following output in the *diagnostics window*:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>-128</td>
</tr>
<tr>
<td>short</td>
<td>-32768</td>
</tr>
<tr>
<td>long</td>
<td>-2147483648</td>
</tr>
<tr>
<td>int</td>
<td>2147483647</td>
</tr>
<tr>
<td>unsigned char</td>
<td>255</td>
</tr>
<tr>
<td>unsigned short</td>
<td>65535</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4294967295</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4294967295</td>
</tr>
</tbody>
</table>
4.2.2 Sample 2 - Defined Data Types (Integers)

Instead of the default data types available in C, specifically defined data types can also be used. These specifically defined data types are however only alias names of the real data types. In this sample, various defined data types are used to display integers. The sample has been configured at the Button2 object displayed below at Event → Mouse → Mouse Action.

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

  CHAR cNumber;  //signed 0 bit value
  SHORT sNumber; //signed 16 bit value
  LONG lNumber;  //signed 32 bit value
  INT iNumber;   //signed 32 bit value
  BOOL bNumber;  //TRUE or FALSE
  BYTE byNumber; //unsigned 8 bit value
  WORD wNumber;  //unsigned 16 bit value
  DWORD dwNumber; //unsigned 32 bit value
  UINT uiNumber; //unsigned 32 bit value

  cNumber = -120;
  sNumber = -32768;
  lNumber = -2147483648;
  iNumber = 2147483647;

  //output in diagnostics window
  printf("\n\nExample 2:\r\n\n");
  printf("CHAR:\t%dx\nSHORT:\t%dx\n"
      "LONG:\t%dx\nINT:\t%dx\n"
     "\n", cNumber, sNumber, lNumber, iNumber);

  bNumber = TRUE;
  byNumber = 255;
  wNumber = 65535;
  dwNumber = 4294967295;
  uiNumber = 4294967295;

  //output in diagnostics window
  printf("\nBOOL:\t%dx\nBYTE:\t%dx\nWORD:\t%dx\n"
      "DWORD:\t%dx\nUINT:\t%dx\n"
    "\n", bNumber, byNumber, wNumber, dwNumber, uiNumber);
}
```

- In the first section, the variables are defined. One variable each of the CHAR, SHORT, LONG and INT defined data type and their unsigned counterparts BYTE, WORD, DWORD and UINT are defined. In addition, a variable of the BOOL data type is defined. Variables of the BOOL data type can be assigned the defined values TRUE or FALSE. The names of the variables are preceded by a prefix describing the data type as in the previous example.
- In the next section, values are assigned to the variables. The number values used in the sample again exactly hit the limits of the value ranges that can be displayed for the various data types.
- The number values are output in the diagnostics window through the function printf(). This output is displayed in the next section.
Output in the Diagnostics Window

The sample described in this section generates the following output in the *diagnostics* window:

Example 2:

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>-128</td>
</tr>
<tr>
<td>SHORT</td>
<td>-32768</td>
</tr>
<tr>
<td>LONG</td>
<td>-2147483648</td>
</tr>
<tr>
<td>INT</td>
<td>2147483647</td>
</tr>
<tr>
<td>BOOL</td>
<td>1</td>
</tr>
<tr>
<td>BYTE</td>
<td>255</td>
</tr>
<tr>
<td>WORD</td>
<td>65535</td>
</tr>
<tr>
<td>DWORD</td>
<td>4294967295</td>
</tr>
<tr>
<td>UINT</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Type Definition

Data types used in this section have been assigned using the `typedef` command. The following program code illustrates how the `BYTE` data type has been defined. `BYTE` is simply the alias name for the default data type `unsigned char` available in C. You can also define your own alias names.

```c
typedef unsigned char BYTE;
```

The following table contains the defined data types pertaining to the default data types available in C:

<table>
<thead>
<tr>
<th>Defined Data Type</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>int</td>
</tr>
<tr>
<td>CHAR</td>
<td>char</td>
</tr>
<tr>
<td>SHORT</td>
<td>short</td>
</tr>
<tr>
<td>LONG</td>
<td>long</td>
</tr>
<tr>
<td>INT</td>
<td>int</td>
</tr>
<tr>
<td>BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>WORD</td>
<td>unsigned short</td>
</tr>
<tr>
<td>DWORD</td>
<td>unsigned long</td>
</tr>
<tr>
<td>UINT</td>
<td>unsigned int</td>
</tr>
</tbody>
</table>
4.2.3 Sample 3 - WinCC Tags (Integers)

In most cases, to solve tasks for making objects dynamic and similar things through a C-Action or another function, WinCC tags must be used. For this purpose, various functions for reading and writing the value of a WinCC tag are available. These functions can be used with each WinCC default tag type. In this sample, values are written to various WinCC tags. The contents of the WinCC tags are displayed in output fields. The sample has been configured at the Button3 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button3

```c
#include <apdefap.h>
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper)
{
   CHAR cNumber; //signed 8 bit value
   SHORT sNumber; //signed 16 bit value
   LONG lNumber; //signed 32 bit value
   BOOL bNumber; //TRUE or FALSE
   BYTE byNumber; //unsigned 8 bit value
   WORD wNumber; //unsigned 16 bit value
   DWORD dwNumber; //unsigned 32 bit value

   cNumber = -120;
   sNumber = -32768;
   lNumber = -2147483648;

   //set wincc tags
   SetTagByte("S001_course_tag_1", cNumber);
   SetTagWord("S161_course_tag_1", sNumber);
   SetTagDWord("S321_course_tag_1", lNumber);

   bNumber = TRUE;
   byNumber = 255;
   wNumber = 65535;
   dwNumber = 4294967295;

   //set wincc tags
   SetTagBitt("BIN1_course_tag_1", (SHORT)bNumber);
   SetTagByte("U001_course_tag_1", byNumber);
   SetTagWord("U161_course_tag_1", wNumber);
   SetTagDWord("U321_course_tag_1", dwNumber);
}
```

- In the first section, the variables are defined. The data types of the variables were selected according to the data types available for WinCC tags.
- In the next section, values are assigned to the variables. The number values used in the sample again exactly hit the limits of the value ranges that can be displayed for the various data types.
- The variable values are assigned to the various WinCC tags using the corresponding functions. The function names consist of the text `SetTag` and the data type designation of the WinCC tag to which the function is applied. Corresponding to the `SetTag` functions for writing to WinCC tags, there are also `GetTag` functions for reading WinCC tags.
• If a variable of the BOOL data type (alias name for int) is transferred to the SetTagBit() function, the compiler will issue a warning. This is done, because the SetTagBit() function expects SHORT as the data type of the transferred variable. Therefore, the content of the bNumber variable is converted to SHORT in this sample code before it is transferred to the SetTagBit() function. This process is also called Typecast (type conversion).

Type Conversion

The content of a variable can be converted to a different data type before it is transferred to a function or assigned to another variable. The data type of the variable itself remains unchanged, however. The following program code illustrates how a variable of the float data type can be converted to the int data type.

    iNumber = (int)fNumber;

Data Types of WinCC Tags

The following table contains the various data types of WinCC tags corresponding to the data types available in C. These are the data types which are transferred to the SetTag functions and returned by the GetTag functions.

<table>
<thead>
<tr>
<th>Data Type of the WinCC Tag</th>
<th>Data Type of the C Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed 8-Bit Value</td>
<td>char</td>
</tr>
<tr>
<td>Signed 16-Bit Value</td>
<td>short int</td>
</tr>
<tr>
<td>Signed 32-Bit Value</td>
<td>long int</td>
</tr>
<tr>
<td>Binary Tag</td>
<td>short int</td>
</tr>
<tr>
<td>Unsigned 8-Bit Value</td>
<td>BYTE</td>
</tr>
<tr>
<td>Unsigned 16-Bit Value</td>
<td>WORD</td>
</tr>
<tr>
<td>Unsigned 32-Bit Value</td>
<td>DWORD</td>
</tr>
</tbody>
</table>
4.2.4 Sample 4 - C Data Types (Floating-Point Numbers)

In this sample, the available default data types of C are used to display floating-point numbers. The sample has been configured at the Button4 object displayed below at Event Mouse → Mouse Action.

C-Action at Button4

```c
#include "apdefap.h"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    float fNumber;  //32 bit
double dNumber;  //64 bit

    fNumber = 1.0000001;
dNumber = 1.0000001;

    //output in diagnostics window
    printf("\r\nExample 4:\r\n");
    printf("float: %.17f\tsizeof(float): %d\r\n", fNumber, sizeof(float));
    printf("double: %.17f\tsizeof(double): %d\r\n", dNumber, sizeof(double));
}
```

- In the first section, the variables are defined. One variable each of the float and double data type is defined.
- In the next section, values are assigned to the variables. In this sample, the same number value is assigned to both variables.
- The accuracy of a variable of the float type goes approximately to the seventh decimal place. A variable of the double type can display numbers twice as accurate. This can be seen in the output of the number values - using the printf() function - in the diagnostics window. In addition to the value of the variable, its memory requirement is also output. The memory requirement of a variable is determined via the sizeof() command. The memory requirement is indicated in Bytes.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 4:
float: 1.00000011920928960  sizeof(float):  4
double: 1.00000010000000010  sizeof(double):  8
4.2.5 Sample 5 - WinCC Tags (Floating-Point Numbers)

In addition to integers, WinCC tags can also contain floating-point numbers. For this, two data types for WinCC tags corresponding to the C data types float and double are available. To access these WinCC tags in read or write mode, corresponding SetTag and GetTag functions are provided. In this sample, values are written to various WinCC tags. The contents of the WinCC tags are displayed in output fields. The sample has been configured at the Button5 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button5

```c
#include "apdefsap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszObjectProp,
{
    float fNumber;  // 32 bit
    double dNumber; // 64 bit

    fNumber = 1.0000001;
    dNumber = 1.0000001;

    // set wincc tags
    SetTagFloat("F32\_course\_tag\_1",fNumber);
    SetTagDouble("F64\_course\_tag\_1",dNumber);
}
```

- In the first section, the variables are defined. One variable each of the float and double data type is defined.
- In the next section, values are assigned to the variables. In this sample, the same number value is again assigned to both variables.
- The variable values are assigned to the various WinCC tags using the corresponding functions. Corresponding to the SetTag functions used here for writing to WinCC tags, GetTag functions for reading WinCC tags are also available.
4.2.6 Sample 6 - Static and External Variables

The sample has been configured at the Button6 object displayed below at Event → Mouse → Mouse Action.

| Count Mouse Clicks |

Static Variables

A C variable is valid in a function after its definition. It becomes invalid again after the function terminates. If the function is called again, the C variable will be generated again. However, if the keyword static precedes the variable, the variable will be maintained in between two function calls. It will therefore keep its value. For C-Actions however, this is only valid as long as the WinCC picture is selected. The static variables become invalid if the picture is deselected. The static variable will be generated again after the picture is opened again during the initial execution of the C-Action.

External Variables

A C variable can only be accessed within the function in which it has been defined. However, if the variable is defined outside of any function, it becomes a global (external) variable. This variable can then be declared in any function using the keyword extern and be accessed.

Project Function CreateExternalTags()

```c
int ext_iNumber = 0;

void CreateExternalTags()
{
    //nothing to do
}
```

- The CreateExternalTags() function only serves the purpose of defining and initializing an external variable of the int type. At the start of the project, the function is called once (at Events →)
- Miscellaneous →
- Open Picture of the start picture cc_0_startpicture_00.PDL. From this time on, the ext_iNumber variable is defined and can be used in any C-Action and any other function.
C-Action at Button6

```c
#include "apdefap.h"
void OnClick(char* lpszPicName, char* lpszObjectName, char* lpszProperty)
{

   // declare external tag
   extern int ext_iNumber;

   // define static tag
   static int stat_iNumber = 0;

   // output in diagnostics window
   printf("%r\nExample 6:\r\n" "mouseclicks since project was started: \d\r\n" "mouseclicks since picture was opened: \d\r\n", ++ext_iNumber, ++stat_iNumber);
}
```

- In the first section, the external variable `ext_iNumber` is declared in order to be able to use the variable in the `C-Action`.
- In the second section, the static variable `stat_iNumber` is defined and initialized. This will be performed at the first execution of the `C-Action` after the WinCC picture has been selected. For further executions of the `C-Action`, the value of the variable will be maintained. If the picture is deselected and then selected again, the variable will be generated again.
- The number values of the variables are incremented by one through the increment operator `++` and output in the `diagnostics window` via the `printf()` function. The variable `ext_iNumber` will therefore indicate the number of clicks on the button since the project start and the variable `stat_iNumber` the number of clicks since the opening of the picture. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the `diagnostics window`:

Example 6:
mouseclicks since project was started: 1
mouseclicks since picture was opened: 1
4.3 Operators and Mathematical Functions in C

In the WinCC project Project_C_Course, samples pertaining to the topic operators can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_01.PDL picture.

Operators

In a program, operators control what happens to variables and constants. Variables and constants are connected to operators - this results in new variable values. Operators can be divided into various categories. This includes mathematical operators, bit-by-bit operators and assignment operators.

Mathematical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ (unary)</td>
<td>Positive sign (actually has no effect)</td>
</tr>
<tr>
<td>- (unary)</td>
<td>Negative sign</td>
</tr>
<tr>
<td>+ (binary)</td>
<td>Addition</td>
</tr>
<tr>
<td>- (binary)</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (returns the remainder of a division)</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
</tr>
</tbody>
</table>
Bit-by-Bit Operators

These operators make it possible to set, query or reset individual bits in variables.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bit-by-Bit AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Bit-by-Bit exclusive OR</td>
</tr>
<tr>
<td>~</td>
<td>Bit-by-Bit inversion</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Move bits to the left</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Move bits to the right</td>
</tr>
</tbody>
</table>

Logical Operators

All logical operators follow the same rule: 0 is FALSE, all other numbers are TRUE. These operators either generate a 0 (FALSE) or a 1 (TRUE).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Logical inversion</td>
</tr>
</tbody>
</table>
4.3.1 Sample 1 - Basic Mathematical Operations

In this sample, the basic mathematical operators are used. The sample has been configured at the Button1 object displayed below at Event ➔ Mouse ➔ Mouse Action.

C-Action at Button1

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    float fValue1 = 123.6;
    float fValue2 = 23.4;
    float fResAdd;
    float fResSub;
    float fResMul;
    float fResDiv;

    fResAdd = fValue1 + fValue2; // add
    fResSub = fValue1 - fValue2; // subtract
    fResMul = fValue1 * fValue2; // multiply
    fResDiv = fValue1 / fValue2; // divide

    // output in diagnostics window
    printf("\nExmple 1\n\n\n%1f + %1f = %1f\n%1f - %1f = %1f\n%1f * %1f = %1f\n%1f / %1f = %1f\n", fValue1, fValue2, fResAdd, fValue1, fValue2, fResSub, fValue1, fValue2, fResMul, fValue1, fValue2, fResDiv);
}
```

- In the first section, two variables of the `float` data type are defined and initialized. The mathematical operators are applied to these two variables.
- In the next section, four additional variables of the `float` data type are defined. These variables store the results of the mathematical operations to be performed.
- In the next section, the mathematical operators are used to add, subtract, multiply and divide.
- The results of these calculations are output through the `printf()` function in the `diagnostics window`. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the `diagnostics window`:

**Example 1**

- `123.6 + 23.4 = 147.0`
- `123.6 - 23.4 = 100.2`
- `123.6 * 23.4 = 2892.2`
- `123.6 / 23.4 = 5.3`
4.3.2 Sample 2 - Increment and Decrement Operator

In this sample, the increment and decrement operators are used. The sample has been configured at the Button2 object displayed below at Event → Mouse → Mouse Action.

Prefix and Postfix

The increment and decrement operators are available in a prefix and postfix version. Both versions perform the same action - they increase or decrease the value of the variable to which they are applied to by one. The difference lies in the return value. The prefix version increments or decrements the value of the variable and returns this new value. The postfix version returns the original value of the variable and only then increments or decrements the variable.

```c
iValue = ++iCount; //prefix
iValue = iCount++; //postfix
```

C-Action at Button2

```c
#include "apdefs.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    static int stat_iPrefix = 0;
    static int stat_iPostfix = 0;
    printf("\r\nExample 2\r\n");
    //execute operators
    printf("Prefix operator on first tag: \%d\r\n", ++stat_iPrefix);
    printf("Postfix operator on second tag: \%d\r\n", stat_iPostfix++);
    //check values
    printf("Value of first tag after execution: \%d\r\n", stat_iPrefix);
    printf("Value of second tag after execution: \%d\r\n", stat_iPostfix);
}
```

- In the first section, two variables of the int data type are defined and initialized.
- The increment operator in the prefix or postfix version are applied to these two variables. The return values of these operators are output through the printf() function in the diagnostics window. Afterwards, the variable contents are also output by the printf() function in the diagnostics window. This output is displayed in the next section.
Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window: It can be seen that the prefix version of the increment operator returns the incremented variable value, whereas the postfix version returns the original variable value. However, in both cases the variable will be incremented.

Example 2
Prefix operator on first tag: 1
Postfix operator on second tag: 0
Value of first tag after execution: 1
Value of second tag after execution: 1
4.3.3 Sample 3 - Bit Operations

In this sample, the basic bit-by-bit operators are used. The sample has been configured at the Button3 object displayed below at Event ➔ Mouse ➔ Mouse Action.

Description

In this sample, the bit-by-bit operators are applied to the content of two WinCC tags (unsigned 16-Bit values). The result of the operation is stored in another WinCC tag of the same type. The operator applied is controlled and simultaneously displayed by the Button6 object. The bit-by-bit connections AND, OR, NAND, NOR and EXOR are available. A number value is assigned to each selection option, which is stored in another WinCC tag (unsigned 8-Bit value).
C-Action at Button3

```c
#include "apndef.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
  BYTE byOperation:
  DWORD dwValue1:
  DWORD dwValue2:
  DWORD dwResult:
  // read tag values
  dwValue1 = GetTagWord("U16i_course_op_1");
  dwValue2 = GetTagWord("U16i_course_op_2");
  // get desired operation
  byOperation = GetTagByte("U08i_course_op_1");
  switch (byOperation)
  {
    case 0: dwResult = dwValue1 & dwValue2;
      break:
    case 1: dwResult = dwValue1 | dwValue2;
      break;
    case 2: dwResult = !(dwValue1 & dwValue2);
      break;
    case 3: dwResult = !(dwValue1 | dwValue2);
      break;
    case 4: dwResult = dwValue1 ^ dwValue2;
      break;
    default: return;
  }
  // write result
  SetTagWord("U16i_course_op_3", (WORD)dwResult);
}
```

- In the first section, one variable of the BYTE data type and three variables of the DWORD data type are defined. These variables are used to temporarily store the WinCC tags.
- In the next section, the two WinCC tags to be connected are read into the dwValue1 and dwValue2 variables. Additionally, the WinCC tag determining the type of the bit-by-bit connection operator will be read into the byOperation variable.
- In the next section, the dwValue1 and dwValue2 variables are connected bit-by-bit depending on the content of the byOperation variable. The connection result is stored in the dwResult variable. The connection operation to be performed is selected by a switch-case construction. This construction is described in greater detail in the Loops chapter.
- In the next section, the connection result contained in the dwResult variable is written to the corresponding WinCC tag.
4.3.4 Sample 4 - Rotating Byte-by-Byte

In this sample, the bit-by-bit move operators are used to rotate the value contained in a WinCC tag (*unsigned 16-Bit value*) byte-by-byte. This means the exchange of the high byte and low byte. The sample has been configured at the *Button4* object displayed below at *Event → Mouse → Mouse Action*.

### C-Action at Button4

```c
#include "apdefsp.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    DWORD dwValue;
    DWORD dwtempValue1;
    DWORD dwtempValue2;

    // read tag value
    dwValue = GetTagWord("U16i_course_op_3");

    // rotate bytes
    dwtempValue1 = dwValue<<8;
    dwtempValue2 = dwValue>>8;
    dwValue = dwtempValue1 | dwtempValue2;

    // write result
    SetTagWord("U16i_course_op_3", (WORD)dwValue);
}
```

- In the first section, a variable of the *DWORD* data type is defined. This variable is used to temporarily store the WinCC tag. In addition, two help variables of the *DWORD* type are defined.
- In the next section, the WinCC tag to be processed is written to the *dwValue* variable.
- In the next section, the individual bits of the *dwValue* variable are moved to the left by eight positions (one byte) and stored in the *dwtempValue1* variable. Afterwards, the individual bits of the *dwValue* variable are moved to the right by eight positions and stored in the *dwtempValue2* variable. Both values determined in this section are connected bit-by-bit (OR) and the result stored in the *dwValue* variable.
- In the next section, the rotated variable value contained in the *dwValue* variable is written to the corresponding WinCC tag.
4.3.5 Sample 5 - Mathematical Functions

In this sample, various mathematical functions are used which are available in C by default. The sample has been configured at the Button5 object displayed below at Event ➔ Mouse ➔ Mouse Action.

C-Action at Button5

```c
#include "appdefs.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    double dValue = 123.6;
    int iValue = -24;
    double dResPow;
    double dResSqrt;
    int iResAbs;
    int iResRand;

dResPow = pow(dValue, 3); // power of 3
dResSqrt = sqrt(dValue); // square root
iResAbs = abs(iValue); // absolute
iResRand = rand(); // random

// output in diagnostics window
printf("Example 5\n\n\n.1f raised to the power of .1f = .1f\n", dValue, dResPow);
printf("Square root of .1f = .1f\n", dValue, dResSqrt);
printf("Absolute value of .1f = .1f\n", iValue, iResAbs);
printf("A pseudorandom number\n\n", iResRand);
}
```

- In the first section, the variables are defined.
- First, the `pow()` function is called. Two parameters are assigned to this function. In this sample, the return value of the function corresponds to the third power of the `dValue` variable content.
- Next, the `sqrt()` function is called. The return value of this function is the square root of the value transferred.
- Next, the `abs()` function is called. The return value of this function is the absolute value of the value transferred.
- Next, the `rand()` function is called. No parameters are assigned to this function. The function will return a random value as the return value.
- The results of these calculations are output through the `printf()` function in the diagnostics window. This output is displayed in the next section.
Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

**Example 5**
123.6 raised to the power of 3 = 1888232.3
Square root of 123.6 = 11.1
Absolute value of -24 = 24
A pseudorandom number = 41

Additional Mathematical Functions

In the function selection, the mathematical functions can be found at *Internal Functions* → *c_bib* → *math*. The following illustration depicts all available mathematical functions (shaded gray).
4.4 Pointers

In the WinCC project Project_C_Course, samples pertaining to the topic pointers can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_02.PDL picture.

![Pointers](image)

Working with Pointers

Pointers are an essential component of the C language. A pointer is a variable containing an address - usually the address of another variable. Pointers are defined just like regular variables. However, the unary character * is added to the name of the data type to which the pointer is pointing. This character must not be mistaken for the binary operator * for the multiplication. In the following program code, a pointer is defined at a variable of the int data type.

```c
int* piValue;
```

The content of the pointer is not defined. It is still pointing to an invalid variable of the int data type. To clarify this, a pointer should be initialized with the value NULL while it is being defined. Before its application, the pointer can then be checked for validity.

```c
int* piValue = NULL;
```

To have the pointer point to a variable of the int data type, it must be assigned the address of the variable. This is done via the unary operator , the so-called address operator. This operator returns the address of the variable instead of its value. In the following program code, the address of a variable with the int data type is assigned to the pointer.

```c
piValue = &iValue;
```

The access to the value of the variable to which the pointer is pointing is realized via the unary operator *, which is also called the content operator. In the following program code, the value of the variable to which the pointer is pointing is assigned to a variable of the int data type.

```c
iValue = *piValue;
```

Working with Vectors

Pointers and vectors are closely related. In the following program code, a vector consisting of 5 variables of the int data type is defined.

```c
int iVector[5];
```
The individual elements of the vectors can be accessed via its index. In the following program code, the content of the last vector element is accessed. This is done via the index operator [ ].

```c
iValue = iVar[4];
```

The vector name can also be used as a pointer pointing to the first vector element. A certain vector element can also be accessed by moving this pointer by a certain number of elements. This is done as illustrated in the program code below by adding a int value to the pointer. The content of the resulting pointer is accessed via the content operator *. As shown previously, the last value of the vector is accessed.

```c
iValue = *(iVar+4);
```

### Character Strings

In C, a character string can be defined as a vector consisting of characters or as a pointer pointing to a character. In addition to the coded character, C adds a null character to the end of the character string. It serves as the end character of the character string. In the program code displayed below, the definition of string variables is shown using both types.

```c
char* lpszString = "String1";
char szString[10] = "String2";
```

Below, the internal display of both string variables is shown. In the first case, exactly as much memory space has been reserved for the string variable as is needed for accepting the string indicated for the initialization. In the second case, as much memory space has been reserved as was specified during the definition of the vector.

<table>
<thead>
<tr>
<th>S</th>
<th>t</th>
<th>r</th>
<th>i</th>
<th>n</th>
<th>g</th>
<th>1</th>
<th>\0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>t</td>
<td>r</td>
<td>i</td>
<td>n</td>
<td>g</td>
<td>2</td>
<td>\0</td>
</tr>
</tbody>
</table>
4.4.1 Sample 1- Pointers

In this sample, the basic pointer operations are performed. The sample has been configured at the Button1 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button1

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperties)
{
    int iValue1 = 126;
    int iValue2 = 23;
    //declare and initialize pointer
    int* piValue = NULL;
    printf("\n\nExample 1\n");
    printf("Address: %x\nValue: undefined\n",piValue);
    //point at iValue1
    piValue = &iValue1;
    printf("Address: %x\nValue: %d\n",piValue,*piValue);
    //point at iValue2
    piValue = &iValue2;
    printf("Address: %x\nValue: %d\n",piValue,*piValue);
}
```

- In the first section, two variables of the int data type are defined and initialized.
- Next, a pointer pointing to a variable of the int data type is defined and initialized with NULL.
- Next, the address contained in the pointer is output via the printf() function. The content, to which the pointer is currently pointing, is not defined. Accessing the content of the pointer via the content operator * would cause a general access violation at this time.
- Next, the address of the iValue1 variable is assigned to the pointer. Its address and content is again output via the printf() function.
- Next, the address of the iValue2 variable is assigned to the pointer and the result output again. The output of this program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Address: 0</th>
<th>Value: undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Address: 2b4f910</td>
<td>Value: 126</td>
</tr>
<tr>
<td></td>
<td>Address: 2b4f91c</td>
<td>Value: 23</td>
</tr>
</tbody>
</table>
4.4.2 Sample 2 - Vectors

In this sample, the basic vector operations are performed. The sample has been configured at the Button2 object displayed below at Event ➔ Mouse ➔ Mouse Action.

C-Action at Button2

```c
#include "apdef.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropName"
{
   // declare and initialize int vector
   int iValue[5] = { 10, 20, 30, 40, 50 };
   int iIndex;
   printf("\r\nExample 2\r\n");
   // access vector elements
   for (iIndex = 0; iIndex<5; iIndex++)
   {
      printf("Index: %d \n Value: %d\r\n", iIndex, iValue[iIndex]);
   }
}
```

- In the first section, a vector consisting of 5 variables of the int data type is defined. The vector is already initialized with number values while it is being defined.
- Next, the Index counter variable of the int data type is defined.
- The individual elements of the vector are output via the `printf()` function. The access to the individual elements is achieved in a for loop via the index operator [ ]. Dealing with loops is described in the next chapter Loops. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

```
Example 2
Index: 0 Value: 10
Index: 1 Value: 20
Index: 2 Value: 30
Index: 3 Value: 40
Index: 4 Value: 50
```
4.4.3 Sample 3 - Pointers and Vectors

In this sample, the relationship between pointers and vectors is explained. The sample has been configured at the Button3 object displayed below at Event ➔ Mouse ➔ Mouse Action.

### C-Action at Button3

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    int iValue[] = { 10, 20, 30, 40, 50 };
    int iIndex;
    int* piElement = NULL;
    printf("\r\nSample 3\r\n");
    //access via separate pointer
    //point to the first element
    piElement = &iValue[0];
    printf("Startaddress: %s\r\n",piElement);
    for (iIndex = 0; iIndex<5; iIndex++)
    {
        printf("Index: %d\tValue: %d\r\n",iIndex,*piElement+iIndex));
    }
    printf("\r\n");
    //access without separate pointer
    printf("Startaddress: %s\r\n",iValue);
    for (iIndex = 0; iIndex<5; iIndex++)
    {
        printf("Index: %d\tValue: %d\r\n",iIndex,iValue+iIndex));
    }
}
```

- In the first section, a vector consisting of 5 variables of the int data type is defined. The vector is already initialized with number values while it is being defined. In this case, the size specification can also be omitted during while defining the vector.
- Next, the iIndex counter variable of the int data type is defined.
- Next, a piElement pointer is defined for a variable of the int data type and initialized with NULL.
- Next, the address of the first vector element is assigned to the piElement pointer. This address is output via the printf() function.
- Next, the individual elements of the vector are accessed by the piElement pointer. The access is carried out in a for loop by advancing the pointer to the individual elements and the content operator *.
• Next, the individual elements of the vector are accessed again. This time, however, the name of the vector itself is used as the pointer. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 3
Startaddress: 2b4f9e8
Index: 0 Value: 10
Index: 1 Value: 20
Index: 2 Value: 30
Index: 3 Value: 40
Index: 4 Value: 50

Startaddress: 2b4f9e8
Index: 0 Value: 10
Index: 1 Value: 20
Index: 2 Value: 30
Index: 3 Value: 40
4.4.4 Sample 4 - Strings

In this sample, working with string variables is explained. The sample has been configured at the Button4 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button4

```c
#include "opcdcf.h"
void OnClick(char* lpwsPictureName, char* lpwsObjectName, char* lpwsRequestParam)
{
    // declare and initialize string
    char szText[13] = "example text";
    int i;
    printf("\r
Example 4:\r
Characters:
[e],[x],[a],[m],[p],[l],[c],[ ],[l],[c],[x],[f].
String:
example text\r
\r
");
    // access single characters
    for (i=0; i<12; i++)
    {
        printf('[%c].', szText[i]);
    }
    printf("\r
");
    // access whole string
    printf("String: \r\n\"szText\":\r
");
}
```

- In the first section, a character string (vector consisting of 13 characters) is defined. This length of the character string is one character more than the length of the assigned initialization string to make room for the closing null character.
- Next, the i counter variable of the int data type is defined.
- Next, the individual characters of the character string are output via the printf() function. The access to these characters is carried out in a for loop via the index operator [ ].
- Next, the entire character string is output via the printf() function. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 4
Characters:
[e],[x],[a],[m],[p],[l],[c],[ ],[l],[c],[x],[f].
String:
example text
4.4.5 Sample 5 - WinCC Text Tags

In this sample, the relation between string variables in C and WinCC text tags is explained. The sample has been configured at the Button5 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button5

```c
#include "apie.h"

void OnClick(char* lpwPictureName, char* lpwObjectName, char* lpwProperty)
{
    //declare and initialize pointer to string
    char* pszText = NULL;

    //get wincc tag value
    pszText = GetTagChar("T081_course_point_1");

    printf("%s\n", pszText);

    //access string
    printf("String: %s\n\nStringlength: %d\nStartaddress: %x\n", pszText, strlen(pszText), pszText);
}
```

- In the first section, a character string (pointer pointing to the first character) is defined. This string is initialized with NULL.
- Next, the content of a WinCC text tag is read in via the `GetTagChar()` function. The function reserves the memory space required for the character string as returns its starting address.
- Next, the entire character string is output via the `printf()` function. In addition, the length of the character string is determined by the `strlen()` function and output together with the starting address of the character string. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

**Example 5**

String: This is an example text!
Stringlength: 24
Startaddress: 1682828
4.5 Loops and Conditional Statements

In the WinCC project Project_C_Course, samples pertaining to the topic loops can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_03.PDL picture.

Loops

Loops can be used to repeatedly perform a code section as long as a condition is satisfied. In general, there are two types of loops: pre-check and post-check loops. The pre-check loops check before the loop body, if this loop is to be performed. The post-check loops check after the loop body, if this loop is to be performed. Therefore, post-check loops are performed at least once.

The following types of loops can be differentiated.

while

A sample of a while loop is displayed below. The loop is repeated as long as the condition is satisfied. In this sample, the loop is performed as long as the value of the i variable is less than 5.

```c
int i = 0;
while (i < 5)
{
   //do something
   ++i;
}
```

do - while

A sample of a do-while loop is displayed below. The loop is performed at least once and then repeated as long as the condition is satisfied. In this sample, the loop is performed as long as the value of the i variable is less than 5.

```c
int i = 0;
do
{
   //do something
   ++i;
} while (i < 5);
```
for

A sample of a for loop is displayed below. The loop is repeated as long as the condition is satisfied. The initialization of the loop counter as well as the processing of the loop counter can be formulated within the loop.

```c
int i = 0;
for (i=0, i<5, i++)
{
    //do something
}
```

Conditional Statements

In loops, the body of the loop is processed for as long as a condition is true. In conditional statements, a statement is processed exactly once if a condition is true.

The following types of conditional statements can be differentiated.

if-else

If the condition is true, the statement in the if branch is processed. If the condition does not apply, the alternative statement in the else branch will be processed. The else branch can also be omitted, if no alternative statement is to be performed.

```c
if (i < 5)
{
    //do something
} else
{
    //do something else
}
```

switch-case

In this case, a variable is checked for a match. Switch specifies the variable to be checked. It is checked, which of the case branches agrees with the value of the variable. This case branch is then performed. Any number of case branches can be defined. Each case branch must end with a break. Optionally, a default branch can be inserted. This branch will be performed, if the value of the variable to be checked agrees with none of the case branches.

```c
switch (i)
{
    case 0: //do something
        break;
    case 1: //do something
        break;
    default: //do something default
        break;
}
```
4.5.1 Sample 1 - while Loop

In this sample, the application of the while loop is explained. The sample has been configured at the Button1 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button1

```c
#include "apclibap.h"
void OnClick(char* lpzzPictureName, char* lpzzObjectName, char* lpzzPointer)
{
    // loop count
    int iCount = 0;
    printf("\nExample 1\n");
    // while loop
    while (iCount < 5)
    {
        // do something
        printf("Executed loop: iCount = %d\n", iCount);
        ++iCount;
    }
    printf("Exit loop: iCount = %d\n", iCount);
}
```

- In the first section, a `iCount` counter variable of the `int` data type is defined and initialized.
- Next, the while loop is programmed. This loop will be executed as long as the content of the `iCount` counter variable is less than 5. Each time the loop is performed, an output is made by the `printf()` function. At the end of the loop, the `iCount` counter variable is increased by one. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 1
Executed loop: iCount = 0
Executed loop: iCount = 1
Executed loop: iCount = 2
Executed loop: iCount = 3
Executed loop: iCount = 4
Exit loop: iCount = 5
4.5.2 Sample 2 - do-while Loop

In this sample, the application of the *do-while* loop is explained. The sample has been configured at the *Button2* object displayed below at *Event* ➔ *Mouse* ➔ *Mouse Action*.

### C-Action at Button2

```c
#include "aple(sp.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    // loop count
    int iCount = 0;
    printf("\r\n\nExample 2\r\n");
    // do-while loop
    do
    {
        // do something
        printf("Executed loop: iCount = %d\r\n", iCount);
        ++iCount;
    } while (iCount < 5);
    printf("Exit loop: iCount = %d\r\n", iCount);
}
```

- In the first section, a *iCount* counter variable of the *int* data type is defined and initialized.
- Next, the do-while loop is programmed. This loop will be executed as long as the content of the *iCount* counter variable is less than 5. However, the loop is performed at least once, since this condition is only checked for after the loop has been performed. Each time the loop is performed, an output is made by the *printf()* function. At the end of the loop, the *iCount* counter variable is increased by one. The output of the program is displayed in the next section.

### Output in the Diagnostics Window

The sample described in this section generates the following output in the *diagnostics window*:

**Example 2**

- Executed loop: iCount = 0
- Executed loop: iCount = 1
- Executed loop: iCount = 2
- Executed loop: iCount = 3
- Executed loop: iCount = 4
- Exit loop: iCount = 5
4.5.3 Sample 3 - for Loop

In this sample, the application of the for loop is explained. The sample has been configured at the Button3 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button3

```c
#include "apdev.h"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperf)
{
    // loop count
    int iCount = 0;
    printf("\nExample 3\n");

    // for loop
    for (iCount=0; iCount<5; iCount++)
    {
        // do something
        printf("Executed loop: iCount = %d\n", iCount);
    }

    printf("Exit loop: iCount = %d\n", iCount);
}
```

- In the first section, a iCount counter variable of the int data type is defined and initialized.
- Next, a for loop is programmed. This loop will be executed as long as the content of the iCount counter variable is less than 5. The initialization of the counter variable is programmed directly in the call of the loop just as the action for incrementing the counter variable. Each time the loop is performed, an output is made by the printf() function. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 3
Executed loop: iCount = 0
Executed loop: iCount = 1
Executed loop: iCount = 2
Executed loop: iCount = 3
Executed loop: iCount = 4
Exit loop: iCount = 5
4.5.4 Sample 4 - Endless Loops

In this sample, endless loops are explained. In most cases, these loops are created unintentionally by programming errors, when a loop condition always holds true. However, they can also be applied intentionally. In this case, the termination of the loop must be realized using another method, namely through the `break` statement. The sample has been configured at the `Button4` object displayed below at `Event → Mouse → Mouse Action`.

C-Action at Button4

```c
#include "apedefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper1
{ // max loop executions
#define MAX_COUNT 1000000

// loop count
int iCount = 0;
int iProgressBar = 1;
char szProgressText[5];

// endless loop
// another possible loop while(TRUE) {...}
for (;;) {
    if (iCount > MAX_COUNT)
    {
        break;
    }
    ++iCount;
    if (iCount-(iProgressBar*MAX_COUNT/100) != 0)
    {
        continue;
    }

    // set value of progress bar
    SetWidth(lpszPictureName,"ProgressBar",(int)(iProgressBar*2.7));

    // set progress text
    sprintf(szProgressText,"%d",iProgressBar);
    SetText(lpszPictureName,"ProgressText",szProgressText);
    ++iProgressBar;
}
```

- In the first section, the symbolic constant `MAX_COUNT` is defined. This constant represents the maximum number of executions for the following endless loop.
- In the next section, a `iCount` counter variable of the `int` data type is defined and initialized.
- The current number of loop executions is to be displayed by a progress display. The display consists of a bar, whose length contains the `iProgressBar` variable and a static text, whose content contains the `szProgressText` string variable.
- Next, the endless loop is programmed. This loop could also be formulated using the `while (TRUE)` statement.
• In the loop, the \emph{iCount} counter variable is checked. If this variable exceeds the value of \emph{MAX_COUNT}, the loop is exited via the \emph{break} statement.
• The \emph{iCount} counter variable will be incremented.
• The progress display shows the loops already performed in percent. For every new percent reached, the value of the progress display is set again. If no new percent has been reached, the loop is immediately performed again via the \emph{continue} statement and the remaining lines skipped.
• The values of the progress display are set by setting the width of the \emph{ProgressBar} bar with the \emph{SetWidth()} function and by setting the text of the \emph{ProgressText} static text with the \emph{SetText()} function. The text used is configured with the \emph{sprintf()} function. This function follows the principle of \emph{printf()}. The text, however, is not output by the Global Script Diagnostics Window, but written to a string variable. This string variable must be defined as the first parameter of the function.
4.5.5 Sample 5 - if-else Statement

In this sample, the application of the if-else statement is explained. The sample has been configured at the Button5 object displayed below at Event → Mouse → Mouse Action.

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
  BYTE byValue;
  //get value to check
  byValue = GetTagByte("U081_course_loop_1");
  printf("\n\nExample 5\n\n");
  if (byValue < 5)
  {
    //do something
    printf("byValue < 5\n");
  }
  else
  {
    //do something
    printf("byValue >= 5\n");
  }
}
```

- In the first section, a byValue variable of the BYTE data type is defined. In this variable, the content of a WinCC tag is stored.
- In the next section, the content of a WinCC tag is read into the byValue variable using the GetTagByte() function.
- Next, a if-else statement is programmed. This statement makes - depending on the content of the byValue variable - an output via the printf() function.
4.5.6 Sample 6 - switch-case Statement

In this sample, the application of the `switch-case` statement is explained. The sample has been configured at the `Button6` object displayed below at Event ➔ Mouse ➔ Mouse Action.

```c
#include "adctrl.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    BYTE byValue;
    //get value to check
    byValue = GetTagByte("U08i_couse_loop_1");
    printf("\n\nSample 6\n\n");
    switch (byValue)
    {
        case 0: //do something
            printf("byValue = 0\n");
            break;
        case 1: //do something
            printf("byValue = 1\n");
            break;
        case 2:
        case 3:
        case 4: //do something
            printf("byValue = 2, 3 or 4\n");
            break;
        default: //do something
            printf("byValue != 0, 1, 2, 3 and 4\n");
            break;
    }
}
```

- In the first section, a `byValue` variable of the `BYTE` data type is defined. In this variable, the content of a WinCC tag is stored.
- In the next section, the content of a WinCC tag is read into the `byValue` variable using the `GetTagByte()` function.
- Next, a `switch-case` statement is programmed. This statement makes - depending on the content of the `byValue` variable - an output via the `printf()` function. To perform the same statements for several different number values of the variable to be checked, the corresponding `case` branches must be arranged among one another. The statements to be performed are programmed in the last `case` branch.
4.6 Functions

In the WinCC project Project_C_Course, samples pertaining to the topic functions can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_05.PDL picture.

Functions

Functions make it possible to better structure a program code. Instead of programming often repeated statements over and over again, they can be shifted into a function. This also results in a central location for editing the program code and easier maintenance. In WinCC, functions can be created as project functions or standard functions.

Transfer Parameters

Values can be transferred to functions, depending on which the function executes statements. These values can be transferred in various ways.

- A constant value can be transferred.
- A variable can be transferred. Only the value of the variable is transferred to the function. The functions has no access to the variable itself.
- A pointer can be transferred. This gives the function access to the variable to which the pointer is pointing. Vectors and structures can only be assigned to a function via pointers.

Return Value

A function can simply execute statements without returning a value. In this case, the return value is of the void data type. If, however, for example a calculation is performed, the value determined can be returned to the caller of the function via the return value. In this case, values or other addresses can be returned. Another option to return values to the caller is to write to a transferred address area. Vectors or structures can only be returned in this manner.
4.6.1 Sample 1 - Transfer of Value Parameters

In this sample, a simple function for calculating the mean value of three numbers is created. The parameters are transferred to the function as values, the result will also be returned as a value. The sample has been configured at the Button1 object displayed below at Event ➔ Mouse ➔ Mouse Action.

Project Function MeanValue()

```c
double MeanValue(double dValue1, double dValue2, double dValue3)
{
    double dMeanValue;
    dMeanValue = (dValue1+dValue2+dValue3)/3;
    return dMeanValue;
}
```

- In the function header, the name of the function is specified as `MeanValue()`. Three variables of the `double` data type are transferred to the function. A variable also of the `double` data type will be returned.
- Next, a variable of the `double` data type, in which the return value will be stored, is defined. This return value is calculated by adding the three transferred values and dividing the resulting sum by three.
- Via the `return` statement, the result is returned to the caller of the function.
C-Action at Button1

```c
#include "apdefsp.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    double dValue1 = 126.2;
    double dValue2 = 23.9;
    double dValue3 = 45.7;
    double dMeanValue;
    //calculate mean value
    dMeanValue = MeanValue(dValue1, dValue2, dValue3);
    //output into diagnostics window
    printf("\nExample 1:\n\nThe mean value of %.1f, %.1f and %.1f = %.1f\n", dValue1, dValue2, dValue3, dMeanValue);
}
```

- In the first section, three variables of the `double` data type are defined and initialized. The mean value of these three tags is to be calculated. An additional variable of the `double` data type is defined which will store the result of the calculation.
- Using the previously created function `MeanValue()`, the mean value of the variables transferred is calculated.
- The result of the calculation is output via the `printf()` function. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the `diagnostics window`:

**Example 1**
The mean value of 126.2, 23.9 and 45.7 = 65.3
4.6.2 Sample 2 - Transfer of Address Parameters

In this sample, a simple function for calculating the mean value of the elements of a vector of any length is created. The address of the vector and its length are transferred to the function. The result is returned as a value. The sample has been configured at the Button2 object displayed below at Event → Mouse → Mouse Action.

Project Function MeanValueVector()

```cpp
double MeanValueVector(double* dValue, DWORD dwSize)
{
    double dSum = 0.0;
    int i;
    for (i = 0; i < dwSize; i++)
    {
        dSum = dSum + dValue[i];
    }
    return (dSum / dwSize);
}
```

- In the function header, the name of the function is specified as `MeanValueVector()`. A pointer pointing to a variable of the `double` data type is transferred to the function. This pointer points to the first element of the vector expected. Additionally, the length of the vector is transferred to the function. A variable of the `double` data type will be returned.
- Next, a variable of the `double` data type is defined and initialized. In this variable, the sum of the elements of the transferred vector are stored. This sum is calculated using a `for` loop.
- Via the `return` statement, the result is returned to the caller of the function. The result corresponds to the sum of the vector elements divided by the number of vector elements.
C-Action at Button2

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    //define and initialize double vector
double dValue[3] = { 128.2, 23.9, 45.7 };
double dMeanValue;
    //calculate mean value of vector
dMeanValue = MeanValueVector(dValue, 3);
    //output into diagnostics window
printf("\nExample 2 \n\n");
printf("The mean value of %.1f, %.1f and %.1f = %.2f\n", 
    dValue[0], dValue[1], dValue[2], dMeanValue);
}
```

- In the first section, a vector consisting of three variables of the *double* data type is defined and initialized. The mean value of these three tags is to be calculated. An additional variable of the *double* data type is defined which will store the result of the calculation.
- Using the previously created function *MeanValueVector()*, the mean value of the transferred vector elements is calculated.
- The result of the calculation is output via the *printf()* function. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the *diagnostics window*:

**Example 2**
The mean value of 128.2, 23.9 and 45.7 = 65.3
4.6.3 Writing to the transferred Address Range

In this sample, a simple function for filling a vector of any length with random numbers is created. The address of a vector and its length are transferred to the function. As the return value, the function will display if the action could be performed successfully via a variable of the BOOL type. The sample has been configured at the Button3 object displayed below at Event → Mouse → Mouse Action.

Project Function FillVector()

```c
BOOL FillVector(int* piVector, DWORD dwSize)
{
    int i;
    // check received pointer
    if (piVector == NULL)
    {
        return FALSE;
    }
    // fill vector
    for (i = 0; i < dwSize; i++)
    {
        piVector[i] = rand();
    }
    return TRUE;
}
```

- In the function header, the name of the function is specified as FillVector(). A pointer pointing to a variable of the int data type is transferred to the function. This pointer points to the first element of the vector expected. Additionally, the length of the vector is transferred to the function. A variable of the BOOL data type is returned indicating whether the function has been performed successfully or not.
- Next, a counter variable of the int data type is defined.
- Next, the transferred pointer is checked. The caller is responsible for the transfer of the correct vector length. If an incorrect value is transferred, this might lead to a general access violation.
- Using a for loop, the elements of the vector transferred are filled by the rand() function with random numbers.
C-Action at Button3

```c
#include "apdefsp.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyID)
{
    #define VECTOR_SIZE 5
    // define int vector
    int iVector[VECTOR_SIZE];
    int i;
    printf("\n\nExample 3\n\n");
    // fill vector
    if (FillVector(iVector, VECTOR_SIZE) == FALSE)
    {
        printf("Error in FillVector\n");
        return;
    }
    printf("Vector Elements: ");
    for (i=0; i<VECTOR_SIZE; i++)
    {
        printf("[%d] ", iVector[i]);
    }
    printf("\n\n");
}
```

- In the first section, a symbolic constant `VECTOR_SIZE` for the number of vector elements is defined.
- Next, a vector `iVector` consisting of `VECTOR_SIZE` variables of the `int` data type is defined.
- Next, a `i` counter variable of the `int` data type is defined.
- Using the previously created `FillVector()` function, the elements of the `iVector` vector transferred are filled with random numbers. The return value of the `FillVector()` function is checked while it is being called with the help of an `if` statement.
- The individual elements of the `iVector` vector are output via the `printf()` function. This output is displayed in the next section.

**Output in the Diagnostics Window**

The sample described in this section generates the following output in the **diagnostics window**:

**Example 3**

Vector Elements: [18467] [6334] [26500] [19169] [15724]
4.6.4 Return of the Result Address

In this sample, a simple function is created which fills a vector random numbers. The length of the desired vector is transferred as a parameter to the function. As the return value, the function will supply the address of the first element of the vector created. The sample has been configured at the Button4 object displayed below at Event → Mouse → Mouse Action.

Project Function GetFilledVector()

```c
int* GetFilledVector(DWORD dwSize)
{
    int* piVector = NULL;
    int i;

    // allocate memory for vector
    piVector = SysAlloc(sizeof(int) * dwSize);

    // check return value of SysAlloc()
    if (piVector == NULL)
    {
        return NULL;
    }

    // fill vector
    for (i=0; i<dwSize; i++)
    {
        piVector[i] = rand();
    }

    return piVector;
}
```

- In the function header, the name of the function is specified as `GetFilledVector()`. The number of elements of the vector created is transferred to the function. The pointer pointing to the first vector element of the `int*` data type is returned.
Next, a `piVector` pointer is defined for a variable of the `int` data type and initialized with `NULL`.

Next, a `i` counter variable of the `int` data type is defined.

Sufficient memory space must be reserved for the vector. This is ensured by the internal function `SysMalloc()`. To this function, the size of the desired memory block is transferred which is calculated by multiplying the memory requirement of a variable of the `int` data type by the desired number of vector elements. The function will return the address of the reserved memory block or `NULL`, if the memory space available is insufficient.

Next, the address received from the `SysMalloc()` function will be checked. If not enough memory space was available, the function is terminated and `NULL` returned.

Using a `for` loop, the elements of the vector are filled by the `rand()` function with random numbers.

The address of the vector created is returned to the caller via the `return` statement.

**C-Action at Button4**

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty, char* lpszPresetName, char* lpszPropertyName)
{
    #define VECTOR_SIZE 6

    //declare pointer to save address of int vector
    int* piVector = NULL;
    int i;
    printf("\n\r\nExample 4\r\n");

    //get address of filled vector
    piVector = GetFilledVector(VECTOR_SIZE);
    if (piVector == NULL)
    {
        printf("Error in GetFilledVector\r\n");
        return;
    }

    printf("Vector Elements: ");
    for (i = 0; i < VECTOR_SIZE; i++)
    {
        printf("%d ,", piVector[i]);
    }
    printf("\n\r\n");
}
```

In the first section, a symbolic constant `VECTOR_SIZE` for the number of vector elements is defined.

Next, a `piVector` pointer is defined for a variable of the `int` data type and initialized with `NULL`.

Next, a `i` counter variable of the `int` data type is defined.

Using the previously created function `GetFilledVector()`, a vector filled with random numbers is created, whose address is then stored in the `piVector` pointer. The return value of the `GetFilledVector()` function is then checked for validity.

The individual elements of the vector created are output via the `printf()` function. This output is displayed in the next section.
Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 4
Vector Elements: [11478] [29358] [25962] [24464] [5705]

Note:
The procedure for transferring structures to functions and the procedure for the return of structures are described in the next chapter.
4.7 Structures

In the WinCC project Project_C_Course, samples pertaining to the topic structures can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_04.PDL picture.

![Structures](image)

**Definition of a Structure Type**

In addition to the default data types, user-defined types can also be defined with the help of structures. In the following program code, a principal structure type definition is displayed. The defined structure type consists of an int and a float structure element. A name must be assigned to each structure element.

```c
struct ExampleStruct
{
    int iElement;
    float fElement;
};
```

**Application of a Structure Variable**

After defining the new structure type, variables of the struct ExampleStruct data type can be defined. This is shown in the following program code. It is also shown how the elements of the structure variable are accessed.

```c
struct ExampleStruct esValue;

esValue.iElement = 100;
esValue.fElement = 0.5;
```

If instead of a structure variable only a pointer pointing to a structure variable is available, its individual elements can be accessed as illustrated by the following program code. Ensure that the pointer is pointing to a valid structure variable or that at least memory space has been reserved for it.
4.7.1 Sample 1 - Structure Variable

In this sample, the definition and application of a simple structure type is explained. The sample has been configured at the StaticText1 object displayed below at Event Mouse Press Left.

C-Action at Static Text1

```c
#include "sphefs.h"
void OnLButtonDown(char* lpzPictureName, char* lpzObjectName, char* lpzObject)
{
    // define structure "CC_POINT"
    struct CC_POINT
    {
        int iLeft;
        int iTop;
    };

    // define structure tag "posObject"
    struct CC_POINT posObject;

    // set structure elements
    posObject.iLeft = x - 8;
    posObject.iTop = y - 8;

    // access structure elements
    SetLeft(lpzPictureName, "cool_man", posObject.iLeft);
    SetTop(lpzPictureName, "cool_man", posObject.iTop);
}
```

- In the first section, a CC_POINT structure type consisting of two int elements is defined. The structure type will accept the coordinates of a mouse click.
- Next, a posObject structure variable of the struct CC_POINT data type is defined.
- Next, values are assigned to the elements of the posObject structure variable. The values assigned are the coordinates of the mouse click. These coordinates are supplied by a C-Action at Event Mouse Press Left as the x and y transfer parameters.
- Next, the coordinates of an object are set with the values contained in the structure variable via the SetLeft() and SetTop() functions.
4.7.2 Sample 2 - Type Definition

In this sample, the definition and application of a simple structure type is explained. As opposed to the previous sample, this structure type will be available in the entire project and not only for one C-Action. The sample has been configured at the Button2 object displayed below at Event ➔ Mouse ➔ Mouse Action.

Structure Definition in apdefap.h

```c
#include "AP_PBIR.H"

//define structure tagCC_Rect
typedef struct tagCC_RECT
{
    int iLeft;
    int iTop;
    int iRight;
    int iBottom;
} CC_RECT; //define type CC_RECT as struct tagCC_Rect
*PCC_RECT; //define type PCC_RECT as pointer to struct tagCC_Rect

//define constants for function GetFileName()
#define GEN_SAVE 0
#define GEN_OPEN 1
```

- A `tagCC_RECT` structure type consisting of four `int` elements is defined. The structure type will accept the position and dimensions of a rectangular area. A variable of this structure type must be defined as a variable of the `struct tagCC_RECT` data type. To avoid this sometimes tedious notation, an alias name is defined using the `typedef` statement. If you now want to define a variable of this data type, it is sufficient to use the `CC_RECT` notation to specify the data type. If a pointer should point to a variable of this data type, the notation `PCC_RECT` can be used.
C-Action at Button2

```c
#include "ap defect.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    //define and initialize CC_RECT structure
    CC_RECT rect = {10, 10, 20, 20};

    //define and initialize pointer to CC_RECT structure
    PCC_RECT prect = NULL;

    printf("\nExample 2\n");

    //access struct elements
    printf("Coordinates: %d, %d, %d, %d\n",
            rect.iLeft, rect.iTop, rect.iRight, rect.iBottom);

    //access struct elements via pointer
    prect = &rect;

    printf("Coordinates: %d, %d, %d, %d\n",
            prect->iLeft, prect->iTop, prect->iRight, prect->iBottom);
}
```

- In the first section, a `rect` variable of the `CC_RECT` data type is defined and initialized.
- Next, a `prect` variable of the `PCC_RECT` data type is defined and initialized with `NULL`. This data type is a pointer pointing to a variable of the `CC_RECT` data type.
- Next, via the operator `.`, the elements of the `rect` structure variable are accessed. Its content is output via the `printf()` function.
- Next, the address of the `rect` variable is assigned to the `prect` pointer. Next, via the operator `->`, the elements of the structure variable - to which the `prect` pointer is pointing - are accessed. The content of the structure variable is again output via the `printf()` function. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Example 2
Coordinates: 10, 10, 20, 20
Coordinates: 10, 10, 20, 20
4.7.3 Sample 3 - WinCC Structure Type

In this sample, the definition and application of a WinCC structure type is explained. Its structure will be identical to the \texttt{CC\_RECT} data type defined in the previous sample. The sample has been configured at the \textit{Button3} object displayed below at \textit{Event Mouse Action}.

WinCC Structure Type Creation

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: WinCC Structure Type Creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A new WinCC structure type is defined in the \textit{WinCC Explorer}. Via a \texttt{R} on the \textit{Structure Types} entry and then selecting \textit{New Structure Type} from the pop-up menu, the dialog for the definition of the properties of a new WinCC structure will be opened.</td>
</tr>
<tr>
<td></td>
<td>![Structure Tag]</td>
</tr>
<tr>
<td></td>
<td>![New Structure Tag...]</td>
</tr>
<tr>
<td></td>
<td>![Find...]</td>
</tr>
<tr>
<td></td>
<td>![Save...]</td>
</tr>
<tr>
<td>2</td>
<td>The dialog \textit{Structure Properties} will be opened.</td>
</tr>
<tr>
<td></td>
<td>The name of the new structure type must be specified. This is done via a \texttt{R} on the default name \textit{NewStructure} and then selecting \textit{Rename} from the pop-up menu. This sample uses the name \textit{Rect}.</td>
</tr>
<tr>
<td></td>
<td>![NewStructure]</td>
</tr>
<tr>
<td></td>
<td>![Delete]</td>
</tr>
<tr>
<td></td>
<td>![Rename]</td>
</tr>
<tr>
<td>3</td>
<td>Definition of the elements of the new structure type.</td>
</tr>
<tr>
<td></td>
<td>A new element can be added via the button \textit{New Element}. The name and the data type of the new element are specified via a \texttt{R} on the element. In the sample, the element is named \textit{Left} and is of the \texttt{LONG} data type. For the element, the radio-button \textit{Internal Tag} is selected. The names and data types of the remaining elements can be seen in the following illustration.</td>
</tr>
<tr>
<td></td>
<td>The \textit{Structure Properties} dialog can be closed by clicking on \textit{OK}.</td>
</tr>
<tr>
<td></td>
<td>![Rect]</td>
</tr>
<tr>
<td></td>
<td>LONG Left</td>
</tr>
<tr>
<td></td>
<td>LONG Top</td>
</tr>
<tr>
<td></td>
<td>LONG Right</td>
</tr>
<tr>
<td></td>
<td>LONG Bottom</td>
</tr>
</tbody>
</table>
Step | Procedure: WinCC Structure Type Creation
---|---
4 | Now WinCC tags of the *Rect* data type can be created. In this sample this means that a total of four tags corresponding to the four structure elements must be created.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>U18i_course_op_1</td>
<td>Unsigned 16-bit value</td>
</tr>
<tr>
<td>U18i_course_op_2</td>
<td>Unsigned 16-bit value</td>
</tr>
<tr>
<td>U18i_course_op_3</td>
<td>Unsigned 16-bit value</td>
</tr>
<tr>
<td>U08i_course_op_1</td>
<td>Unsigned 8-bit value</td>
</tr>
</tbody>
</table>

C-Action at Button3

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    //define CC_RECT structure
    CC_RECT rect;
    //read element values of wincc structure tag
    rect.iLeft = GetTagSWord("STRI_course_str_1.Left");
    rect.iTop = GetTagSWord("STRI_course_str_1.Top");
    rect.iRight = GetTagSWord("STRI_course_str_1.Right");
    rect.iBottom = GetTagSWord("STRI_course_str_1.Bottom");
    printf("\nExample 3\n");
    //access struct elements
    printf("Coordinates: %d, %d, %d, %d\n",
           rect.iLeft, rect.iTop, rect.iRight, rect.iBottom);
}
```

- In the first section, a *rect* variable of the *CC_RECT* data type is defined. The *CC_RECT* data type has been defined in the previous sample.
- Next, the values contained in a WinCC structure tag are written to the elements of the *rect* variable. In this sample, the values contained in the WinCC structure tag are displayed via four *I/O Fields* and can also be edited in them.
- Next, the elements of the *rect* structure variable are output via the *printf()* function. The output of the program is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the *diagnostics window*:

**Example 3**

**Coordinates: 0, 0, 0, 0**
4.7.4 Sample 4 - Function for Reading a WinCC Structure Type

In this sample, a function for reading the WinCC structure type defined in the previous sample is created. This function can then be used similarly like the internal GetTag functions. The sample has been configured at the Button4 object displayed below at Event Mouse → Mouse Action.

Project Function GetTagRect()

```c
#include "apdefap.h"

void* GetTagRect(char* lpszTagName) {
    PCC_RECT prect = NULL;
    // max size of struct tag name = 260
    // max size of element name = 7
    char szElementTag[267];  // 260 + 7
    // allocates memory for CC_RECT structure
    prect = SysMalloc(sizeof(CC_RECT));
    // check return value of SysMalloc()
    if (prect == NULL) {
        return NULL;
    }

    /////////////////////////////////////////////////////////////////////////////
    // create tag names and set tag values
    sprintf(szElementTag, "%s Left", lpszTagName);
    prect->iLeft = GetTagSWord(szElementTag);

    sprintf(szElementTag, "%s Top", lpszTagName);
    prect->iTop = GetTagSWord(szElementTag);

    sprintf(szElementTag, "%s Right", lpszTagName);
    prect->iRight = GetTagSWord(szElementTag);

    sprintf(szElementTag, "%s Bottom", lpszTagName);
    prect->iBottom = GetTagSWord(szElementTag);

    /////////////////////////////////////////////////////////////////////////////

    // return address of structure as void*
    return (void*) prect;
}
```

- In the first section, the `apdefap.h` file is integrated, which contains the definition of the `tagCC_RECT` structure type.
- In the function header, the name of the function is specified as `GetTagRect`. A string variable is transferred to the function containing the name of the WinCC structure tag to be read. A pointer pointing to a memory block of an undefined data type (`void*`) is returned.
- In the next section, a `prect` variable of the `PCC_RECT` data type is defined and initialized with `NULL`. This data type is a pointer pointing to a variable of the `CC_RECT` data type.
• Next, a string variable for accepting the element names of the WinCC structure tag is created.
• Next, enough memory space must be reserved to accept a variable of the `CC_RECT` data type. This is ensured by the internal function `SysMalloc()`. The size of the desired memory block, which can be determined via the `sizeof()` statement, is transferred to this function. The function will return the address of the reserved memory block or `NULL`, if the memory space available is insufficient.
• Next, the address received from the `SysMalloc()` function will be checked. If not enough memory space was available, the function is terminated and `NULL` returned.
• Next, the names of each element of the WinCC structure tag are composed and the content of the corresponding element read into the reserved memory area.
• The address of the reserved memory block, in which the content of the WinCC structure tag has been stored, is returned. This memory block will remain reserved and keep its data even after the function has been exited.

**Note:**
If, instead of the procedure presented here, simply a local variable of the `CC_RECT` data type had been created - whose elements contain the content of the WinCC structure tag and returns its address - the caller of the function would receive an invalid pointer. This can be explained by the fact that the variable of the `CC_RECT` data type would become invalid at the end of the function and consequently an address of an invalid object would be returned.
C-Action at Button4

```c
#include "opcode.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    // define and initialize pointer to PCC_RECT structure
    PCC_RECT prect = NULL;

    // read element values of wincc structure tag
    prect = (PCC_RECT) GetTagRect("STR1_course_str1");
    printf("\n\nExample 4\n\n");
    // check return value of GetTagRect()
    if (prect == NULL)
    {
        printf("Error in GetTagRect()\n");
        return;
    }

    // access struct elements
    printf("Coordinates: %d, %d, %d, %d\n",
           prect->iLeft, prect->iTop, prect->iRight, prect->iButton);
}
```

- In the first section, a `prect` variable of the `PCC_RECT` data type is defined and initialized with `NULL`.
- Next, a WinCC structure tag is read via the previously created `GetTagRect()` function. The `GetTagRect()` function returns a pointer pointing to the memory block containing the desired data. This pointer is converted to a pointer of the `PCC_RECT` type.
- Next, the pointer received from the `GetTagRect()` function is checked. If not enough memory space is available, the function will return the value `NULL`.
- Next, the names of each element of the WinCC structure tag are composed and the content of the corresponding element read into the reserved memory area.
- The address of the reserved memory block, in which the content of the WinCC structure tag has been stored, is returned. This memory block will remain reserved and keep its data even after the function has been exited.
- Next, the elements of the structure tag, to which `prect` is pointing, are output via the `printf()` function. The output of the program is displayed in the next section.

**Output in the Diagnostics Window**

The sample described in this section generates the following output in the *diagnostics window*:

*Example 4*

*Coordinates: 0, 0, 0, 0*

**Project Function SetTagRect()**

In addition to the `GetTagRect()` function, a corresponding `SetTagRect()` function has been created as well. In the sample, this function is used at **Event ➔ Miscellaneous ➔ Open Picture** of the `cc_9_example_04.PDL` picture to initialize the WinCC structure tag.
4.8 WinCC API

In the WinCC project Project_C_Course, samples pertaining to the topic WinCC API can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_10.PDL picture.

WinCC Application Program Interface

As a completely open and expandable system, WinCC provides an extensive API (Application Program Interface). This is an interface, which is used by applications to access WinCC. The functions of the WinCC API can also be used in the WinCC project itself.

The WinCC ODK (Open Developers Kit) provides an extensive description of the WinCC API. In it, the WinCC API is explained thoroughly by means of function descriptions and samples. It also includes all header files and required function declarations. However, the WinCC ODK is not part of the WinCC base package and must be purchased separately.
Function Libraries

Each (major) application of WinCC (Graphics Designer, Tag Logging, Alarm Logging, etc.) provides its own API, which is located in one or multiple DLLs. A DLL (Dynamic Load Library) is a dynamically loaded function library. The declarations of the functions contained in a DLL are provided in an associated header file. The integration of a DLL into a C-Action or other function is shown in the following program code sample. In the first line, the name of the DLL to be loaded is specified. In the sample, this is the DLL which contains the CS functions of the Graphics Designers. In the second line, the header file with the function declarations is integrated. If only one or two functions are required, the declaration of the function can also be made directly at this point. The closing line is formed by #pragma code(). In the above sample, the names of the DLL and header files agree, which appears to make sense. This, however, is not always the case.

```c
#pragma code("PDLCSAPI.dll")
#include 'pdlcsapi.h'
#pragma code()
```

RT Functions and CS Functions

The API functions of every application can roughly be separated into two different types. These are the so-called CS functions (Configuration System) and RT function (Runtime). In most cases, RT functions can be called without the loading of a particular DLL in a WinCC project. RT functions only affect runtime. After the restart of a project or in most cases even after a picture change, modifications made using RT functions will be lost. Before the application of a CS function in a WinCC project, the corresponding DLL - in which the function has been programmed - must be loaded. The application of CS functions in the WinCC project itself, however, only makes sense in rare cases. The present sample nevertheless will illustrate the application of CS functions, since from, the basic principle of its application can be deduced for your own projects.

Sample Project

In the sample project, no detailed description of every WinCC application API is provided. However, the general principles of working with the WinCC API are explained, using the Graphics Designers API as an example. The samples work with objects in the cc_9_example_10x.PDL picture, especially configured for this purpose. It is displayed via a Picture Window in the picture assigned to this chapter.
4.8.1 Sample 1 - Changing Properties via an RT Function

In this sample, the application of the RT function of the Graphics Designer API for setting the properties of objects is shown. The position of an object is changed by setting the properties Position X and Position Y. The sample has been configured at the Button1 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button1

```c
#include "apndef.h"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropName)
{
    BOOL bRet = FALSE;

    // picture name without extension ' PDL'
    char szPictureName[] = "cc_9_example_1Pdl";
    char szObjectName[] = "I/OField1";

    // property type
    VARTYPE vt = VT_I4;

    // new property values
    int iValueLeft = 60;
    int iValueTop = 70;

    // error structure
    CW_ERROR Error;

    // set the property and check the return values
    bRet = PDLYSetPropEx(0, szPictureName, szObjectName, "Left", vt, iValueLeft, NULL, NULL, 0, NULL, &Error);
    if (bRet == FALSE) {
        print("\nError in PDLYSetPropEx()\n" "\tError.szErrorText");
    }
    bRet = PDLYSetPropEx(0, szPictureName, szObjectName, "Top", vt, iValueTop, NULL, NULL, 0, NULL, &Error);
    if (bRet == FALSE) {
        print("\nError in PDLYSetPropEx()\n" "\tError.szErrorText");
    }
```

- In the first section, a `bRet` variable of the `BOOL` data type is defined and initialized. This variable will accept the return value of the API functions called.
- Next, two string variables are defined. Their content - the picture name and object name - specifies the object to be edited. Make sure that the picture name does not include the file extension `PDL`.
- To define the type of the properties to be set, a variable is created. For this, a separate data type is available. This is the `VARTYPE` data type. For each existing property type, a symbolic constant is defined. The properties to be set for this sample are of the `VT_I4` data type (`int` with a length of 4 Bytes).
For the Position X and Position Y properties to be set, a variable each of the int type is defined, which includes the new value of the property.

Next, a variable of the CMN_ERROR data type is defined. If a function call fails, this structure will contain information about the error occurred.

Via the PDLRTSetPropEx() API function, the Position X and Position Y properties of the specified object are set. The first parameter of the API function denotes the addressing mode of the object. The next three parameters denote the desired picture name, object name and property name. To specify the desired property, the English name, not the German one, must be used. For the previous example, these are Left and Top. The next parameter denotes the property type. In the following parameter, the address of the variable is specified, which contains the new value of the property. The next four parameters are not relevant for the desired functionality. In the last parameter, the address of the error structure is specified.

The return value of every API function is checked in an if statement after it has been called. If the call fails, this is indicated by an output. Part of this output includes the information contained in the szErrorText structure element of the error structure.

Note:
In this sample, the return value of the API function is assigned to a variable of the BOOL data type. This variable is then checked. The call of the API function and the check of the return value can also be combined into one line. This procedure will be used in the subsequent samples of this chapter.
4.8.2 Sample 2 - Creating a Tag Connection via an RT Function

In this sample, the application of the RT function of the Graphics Designer API for creating a tag connection is shown. A tag connection is created at Property → Output/Input → Output Value of an I/O Field. The sample has been configured at the Button2 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button2

```c
#include "apdefs.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    //include file with the LinkType definitions
#include "trigger.h"

    char szPictureName[] = "cc_9_example_10ex";
    char szObjectName[] = "I/OField1";

    LINKINFO link;
    CMN_ERROR Error;

    //fill link info structure
    link.LinkType = BUBRT_LT_VARIABLE_DIRECT;
    link.dwCycle = 0;
    strcpy(link.szLinkName,"U081_course_wincc_2");

    //set link and check the return value
    if ( PDERTSetLink(0,szPictureName,szObjectName,"OutputValue",
            &link,NULL,NULL,&Error) == FALSE) {
        printf("\n\nError in PDERTSetLink()\n\n%\n\n", Error.szErrorMessage);
        printf("\n\n%\n\n", Error.szErrorText);
    }
}
```

- In the first section, the trigger.h file is integrated. This file contains the definition of a symbolic constant used in this sample.
- Next, two string variables are defined. Their content - the picture name and object name - specifies the object to be edited.
- To specify the properties of a tag connection, a separate data type is available. This is the LINKINFO structure type. A link variable of the LINKINFO data type is defined.
- Next, a variable of the CMN_ERROR data type is defined.
- The structure elements of the link variable are filled with the information of the desired tag connection. The LinkType element is assigned the BUBRT_LT_VARIABLE_DIRECT symbolic constant. This constant stands for a direct tag connection. The dwCycle element is assigned the value 0, which corresponds to the Upon Change trigger. The szLinkName element specifies the variable to be used.
• Via the `PDLRTSetLink()` API function, a Tag Connection at the specified object is created. The first parameter of the API function denotes the addressing mode of the object. The next three parameters denote the desired picture name, object name and property name. In the following parameter, the address of the `link` variable is specified, which determines the tag connection to be created. The next two parameters are not relevant for the desired functionality. In the last parameter, the address of the error structure is specified. If the call of the API function fails, this is indicated by an output.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first two samples of this chapter refer to the <code>I/OField1</code> object of the <code>cc_9_example_10x.PDL</code> picture. The first sample changes the position of the object in the picture, the second creates a tag connection at it. After a picture change, the modifications made are lost. Ensure that after the execution of the CS functions described in the subsequent samples, a picture change must always be performed. This means that changes in the picture achieved by using one of the first two button will be lost after clicking on one of the other buttons.</td>
</tr>
</tbody>
</table>
4.8.3 Sample 3 - Creating a New Object via a CS Function

In this sample, the application of the CS function of the Graphics Designer API for creating a new object is shown. A new I/O Field will be created. The sample has been configured at the Button3 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button3

```c
#include "apdocap.h"
void OnClick(char* lpszAppName, char* lpszObjectName, char* lpszProperty);

#define PDLCSSAPI "PDLCSSAPI.dll"
#include "pdllcppapi.h"

char szProjectName[MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL"; //with ".PDL"
DWORD dwFlag = 1; //do not display picture
CHN_ERROR Error;
char szObjectName[] = "1/OField";
GUID guid = GUID_I0Field; //object type i/o field

//get project name
if ( DNGetsRuntimeProject(szProjectName,MAX_PATH+1,&Error) == FALSE )
{
    printf("\r\n\r\nError in DNGetsRuntimeProject()\r\n\r\n\r\n".Error.szErrortext);
    return;
}

//initialize API interface of the graphics designer
if ( PDLCSSGetOleAppPtr(FALSE,&Error) == FALSE )
{
    printf("\r\n\r\nError in PDLCSSGetOleAppPtr()\r\n\r\n".Error.szErrortext);
    return;
}

//open picture without displaying it
if ( PDLCSSOpen(szProjectName,szPictureName,dwFlag,&Error) == FALSE )
{
    printf("\r\n\r\nError in PDLCSSOpen()\r\n\r\n".Error.szErrortext);
    goto OPEN_FAILED;
}

//create object
if ( PDLCSSNewObjectEx(szProjectName,szPictureName,&guid,
                        szObjectName,&Error) == FALSE )
{
    printf("\r\n\r\nError in PDLCSSNewObjectEx()\r\n\r\n".Error.szErrortext);
    goto ACTION_FAILED;
}

//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE )
{
    printf("\r\n\r\nError in PDLCSSave()\r\n\r\n".Error.szErrortext);
    goto ACTION_FAILED;
}

//actualize the picture which contains the created object
ActualizeObject();

//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED: PDLCSSDtlOleAppPtr(FALSE);
```

In the first section, the DLL of the Graphics Designer API is loaded. In addition, the pdl_guid.h file is integrated, which contains the definition of a symbolic constant used in this sample.

Next, a szProjectName string variable for accepting the project name is defined.

Next, a string variable for accepting the picture name is defined. Note that the picture name must include the file extension PDL (which differs from the RT functions).

Next, the additional variables required are defined. This includes a variable of the GUID data type, which determines the object type to be created.

In the next section, the project name is determined via the DMGetRuntimeProject() API function. The project name will be stored in the szProjectName variable. If the determination of the project name fails, this will will be indicated by an output. In this case, the C-Action is exited via the return statement.

In the next section, the Graphics Designer API is initialized by the PDLCSGetOleAppPtr() API function. If the initialization of the Graphics Designer API fails, this will will be indicated by an output. In this case, the C-Action is exited via the return statement as well.

In the next section, the picture to be edited is opened via the PDLCSOpenEx() API function. As the next to last parameter, the dwFlags variable set to the value 1 is transferred to the API function. This causes the picture not to be displayed on the screen. If the opening of the picture fails, this will will be indicated by an output. Via the goto statement, a jump is made to the code position where the connection to the Graphics Designer API is terminated.

In the next section, a new object named I/OField2 will be created via the PDLCSNewObjectEx() API function. If the creation of the new object fails, this will will be indicated by an output. Via the goto statement, a jump is made to the code position where the previously opened picture is closed.

In the next section, the picture is saved via the PDLCSSave() API function. If saving the picture fails, this will will be indicated by an output. Via the goto statement, a jump is made to the code position - just as in the section above - where the previously opened picture is closed.

Next, the picture to be edited is selected again via the project function ActualizeObjects().

Next, the previously opened picture is closed again via the PDLCSClose() API function. Before this statement, a mark is inserted, which is the jump target for previous goto statements.

Next, the connection to the Graphics Designer API is terminated again via the PDLCSDelOleAppPtr() API function. Before this statement, a mark is inserted as well, which is the jump target for a previous goto statement.

Note:
The C-Actions, which will be created in the subsequent samples are very similar to the C-Action created in the present sample. We will therefore omit detailed explanations, which have been provided in the present sample. The code descriptions will be limited to an overview of the program’s execution.
4.8.4 Sample 4 - Changing Properties via a CS Function

In this sample, the application of the CS function of the Graphics Designer API for setting the properties of objects is shown. The position of an object is changed by setting the properties Position X and Position Y. The sample has been configured at the Button4 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button4

```c
#include "optdefsp.h"
void OnClick(char* lpszObjectName, char* lpszObjectName, char* lpszPropertyName)
{
//pragma code("POLCSPA dance")
#include "polcspsi.h"
#endif

char szObjectName[MAX_PATH];
char szPictureName[] = "cc_4_example_10ex.PDF";
char szObjectName[] = "L/OField1"
char szPropertyName[2][5] = { "Left", "Top" }

.elementAt = VT_ID4;
int iValue[] = { 60, 130 }
int i
CMH_ERROR Error;

//get project name
if (DMGetRuntimeProject(szObjectName, MAX_PATH+1,&Error) == FALSE)
{
    printf("\n\nError in DMGetRuntimeProject()\n\n". Error.szErrorText);
    return;
}

//initialize API interface of the graphics designer
if (PDCSGetOleAppPtr(FALSE,&Error) == FALSE)
{
    printf("\n\nError in PDCSGetOleAppPtr()\n\n". Error.szErrorText);
    return;
}

//open picture without displaying it
if (PDCSOpenEx(szObjectName, szPictureName, 1,&Error) == FALSE)
{
    printf("\n\nError in PDCSOpenEx()\n\n". Error.szErrorText);
    goto OPEN_FAILED;
}

//set properties
for (i=0; i<2; i++)
{
    if (PDCSSetPropertyEx(szObjectName, szPictureName, szObjectName,
               szPropertyName[i], vt.iValue[1], 0, NULL,&Error) == FALSE)
    {
        printf("\n\nError in PDCSSetPropertyEx()\n\n". Error.szErrorText);
    }
}

//save picture
if (PDCSSave(szObjectName, szPictureName, &Error) == FALSE)
{
    printf("\n\nError in PDCSSave()\n\n". Error.szErrorText);
    goto ACTION_FAILED;
}

//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDCSClose(szObjectName, szPictureName, &Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED: PDCSUnlOleAppPtr(FALSE);
```

• In the first section, the DLL of the Graphics Designer API is loaded.
• In the second section, the required variables are defined. The names of the properties and the property values to be set are stored in vectors, as opposed to following the procedure described in sample 1.
• The project name is determined via the DMGetRuntimeProject() API function.
• The Graphics Designer API is initialized by the PDLCSGetOleAppPtr() API function.
• The picture to be edited is opened via the PDLCSOpenEx() API function.
• Within a for loop, the properties of the object are set via the PDLCSSetPropertyEx() API function. If properties of different types are to be set in this way, a vector must be defined instead of the vt variable. This vector will determine the property type of each property to be set.
• The picture is saved via the PDLCSSave() API function.
• The picture to be edited is selected again via the project function ActualizeObjects().
• The picture is closed again via the PDLCSClose() API function.
• The connection to the Graphics Designer API is terminated again via the PDLCSDelOleAppPtr() API function.
4.8.5 Sample 5 - Creating a Tag Connection via a CS Function

In this sample, the application of the CS function of the Graphics Designer API for creating a tag connection is shown. A tag connection is created at Property ➔ Output/Input ➔ Output Value of an I/O Field. The sample has been configured at the Button5 object displayed below at Event ➔ Mouse ➔ Mouse Action.

C-Action at Button5

```c
#include "apdefs.h"
void OnClick(char* lpszObjectName, char* lpszObjectName, char* lpszProperty)
{  
    char szObjectName[MAX_PATH];  
    char szClassName[] = "cc_9_example10ex.PDF";  
    char szObjectPath[] = "I/OField2";
    LINKINFO link;
    CMN_ERROR Error;

    //get project name
    if (DNGetRuntimeProject(szObjectName, MAX_PATH+1,&Error) == FALSE)
    {
        printf("\r\nError in DNGetRuntimeProject()\r\n""\t Error.szErrorText"");
        return;
    }

    //initialize API interface of the graphics designer
    if (PDLCSSetOleAppPtr(TRUE,&Error) == FALSE)
    {
        printf("\r\nError in PDLCSSetOleAppPtr()\r\n""\t Error.szErrorText"");
        return;
    }

    //open picture without displaying it
    if (PDLCSSave(szObjectName, szClassName+1,&Error) == FALSE)
    {
        printf("\r\nError in PDLCSSave()\r\n""\t Error.szErrorText"");
        goto ACTION_FAILED;
    }

    //set link info struct
    link.cmLinkType = DBRT_LT_VARIABLE_DIRECT;
    link.dwCycle = 0;
    strcpy(szLinkName,"H081_cource_winc_c_1."");

    //set link
    if (PDLCSSetLink(szObjectName, szClassName, szObjectPath, "OutputValue", 
        &link, &Error) == FALSE)
    {
        printf("\r\nError in PDLCSSetLink()\r\n""\t Error.szErrorText"");
        goto ACTION_FAILED;
    }

    //save picture
    if (PDLCSSave(szObjectName, szClassName, &Error) == FALSE)
    {
        printf("\r\nError in PDLCSSave()\r\n""\t Error.szErrorText"");
        goto ACTION_FAILED;
    }

    //actualize the picture which contains the created object
    ActualizeObjects();

    //close picture
    ACTION_FAILED: PDLCSClose(szObjectName, szClassName, &Error);
    //disconnect from the API interface of the graphics designer
    OPEN_FAILED: PDLCSSetOleAppPtr(FALSE);
}
```
In the first section, the DLL of the Graphics Designer API is loaded.
In the second section, the required variables are defined.
The project name is determined via the DMGetRuntimeProject() API function.
The Graphics Designer API is initialized by the PDLCSGetOleAppPtr() API function.
The picture to be edited is opened via the PDLCSOpenEx() API function.
In the next section, the structure elements of the link variable are filled with the information of the desired tag connection.
The tag connection at the object is created via the PDLCSSetLink() function.
The picture is saved via the PDLCSSave() API function.
The picture to be edited is selected again via the project function ActualizeObjects().
The picture is closed again via the PDLCSClose() API function.
The connection to the Graphics Designer API is terminated again via the PDLCSDelOleAppPtr() API function.
4.8.6 Sample 6 - Listing Objects via a CS Function

In this sample, the application of the CS function of the Graphics Designer API for listing objects contained in a picture is shown. For each available object, the API will call an especially created function to which the information about the corresponding object is transferred. Such a function is called a Callback function. The sample has been configured at the Button6 object displayed below at Event → Mouse → Mouse Action.

Project Function ObjectCallback()

```c
#include "pdclsapi.h"
//include file with OBJECT_INFO_STRUCT definition

BOOL ObjectCallback(void* lpData, void* item)
{
    //pointer to OBJECT_INFO_STRUCT
    LPOBJECT_INFO_STRUCT lpInfoStruct = NULL;
    //store received address of OBJECT_INFO_STRUCT
    lpInfoStruct = (LPOBJECT_INFO_STRUCT) lpData;
    //check received address
    if (lpInfoStruct == NULL)
    {
        printf("Error in ObjectCallback()\n");
        return FALSE;
    }
    //access structure element
    printf("%s\n", lpInfoStruct->szObjectName);
    return TRUE;
}
```

- In the first section, the pdclsapi.h file is integrated, which contains the definition of the OBJECT_INFO_STRUCT structure type.
- The data type of the return value as well as the data types and the quantity of transfer parameters are specified for this function and can be obtained from the WinCC ODK.
- Next, a pointer pointing to a variable of the OBJECT_INFO_STRUCT data type is defined. To this pointer, the address contained in the first transfer parameter is assigned. The pointer is then checked for validity.
- The name of the object, whose OBJECT_INFO_STRUCT was received, will be output.
C-Action at Button6

```c
#include "apdefap.h"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyp
{
    #pragma code("PDLCSAPI.dll")
    #include "pdlcssapi.h"
    #pragma code()

    char szProjectName[MAX_PATH];
    char szPictureName[] = "cc_9_example_10ex.FDL";
    CNH_ERROR Error;

    // get project name
    if (DMGetRuntimeProject(szProjectName, MAX_PATH, &Error) == FALSE)
    {
        printf("\r\nError in DMGetRuntimeProject()\r\n" "%s\r\n", Error.szErrorText);
        return;
    }

    // initialize API interface of the graphics designer
    if (PDLCSGetOleAppPtr(FALSE, &Error) == FALSE)
    {
        printf("\r\nError in PDLCSGetOleAppPtr()\r\n", Error.szErrorText);
        return;
    }

    // open picture without displaying it
    if (PDLCSOpenEx(szProjectName, szPictureName, 1, &Error) == FALSE)
    {
        printf("\r\nError in PDLCSOpenEx()\r\n", Error.szErrorText);
        goto OPEN_FAILED;
    }

    printf("\r\nObjects in picture cc_9_example_10ex.FDL\r\n");

    // enumerate objects
    if (PDLCSEnumObject(szProjectName, szPictureName, ObjectCallback, NULL, &Error) == FALSE)
    {
        printf("\r\nError in PDLCSEnumObject()\r\n", Error.szErrorText);
    }

    // close picture
    PDLCSClose(szProjectName, szPictureName, &Error);

    // disconnect from the API interface of the graphics designer
    OPEN_FAILED: PDLCSDelOleAppPtr(FALSE);
}
```

- In the first section, the DLL of the Graphics Designer API is loaded.
- In the second section, the required variables are defined.
- The project name is determined via the DMGetRuntimeProject() API function.
- The Graphics Designer API is initialized by the PDLCSGetOleAppPtr() API function.
- The picture to be edited is opened via the PDLCSOpenEx() API function.
- All objects contained in the picture will be listed by the PDLCSEnumObjList() API function. For this purpose, the address of the previously created ObjectCallback() project function is transferred to the API function. This type of function is also known as a Callback function. It will be called once for every object contained in the picture, at which point the data of one object will be transferred.
- The picture is closed again via the PDLCSClose() API function.
- The connection to the Graphics Designer API is terminated again via the PDLCSDelOleAppPtr() API function.
Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

Objects in picture cc_9_example_10ex.PDL:
cc_9_example_10ex
I/OField1
I/OField2
4.9 Project Environment

In the WinCC project Project_C_Course, samples pertaining to the topic Project Environment can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_11.PDL picture.

General Information

Many times the programming of a C-Action or other function requires the specification of a file path, the name of the local computer and such. These values can then be specified - according to the current conditions - as absolute values. This can cause problems if a project is transferred to another computer. The conditions encountered there are in all likelihood different from those of the creation system. It is therefore recommended to refrain from using absolute path specifications and such when creating a project. Instead such information should be determined while in runtime. This chapter contains samples, which illustrate how information about conditions on the local computer are accessed. For this purpose, the WinCC API as well as the Windows API are used.
4.9.1 Sample 1- Determining the Project File

In this sample, the procedure for determining the project file of a WinCC project is outlined. The sample has been configured at the Button1 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button1

```c
#include "apdevap.h"
void OnClick(char* lpFileName, char* lpObjectName, char* lpObjectName)
{
    BOOL bRet;
    char szProjectFile[_MAX_PATH+1];
    CMN_ERROR Error;
    //get the project file name
    bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);
    //check return value
    if (bRet == FALSE)
    {
        printf("\n\nError in DMGetRuntimeProject()\n" "\n\"%s\",Error.szErrorText);
        return;
    }
    //display project file
    printf("\n\nProject File:\r\n\r\n\r\n".szProjectFile).
}
```

- In the first section, a bRet variable of the BOOL data type is defined.
- Next, a szProjectFile string variable for accepting the project name is defined. The length of this tag is set to accommodate (store) the longest possible path specification.
- Next, a variable of the CMN_ERROR data type is defined.
- Next, the project name is determined via the DMGetRuntimeProject() API function. The project name will be stored in the szProjectFile variable. As the second parameter, the size of the memory space reserved for the project name is specified. As the third parameter, the address of the error structure is specified. If no error information is needed, NULL can be transferred.
- Next, the return value of the DMGetRuntimeProject() API function is checked.
- Next, the determined name of the project file is output. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

```plaintext
ProjectFile: \WINCC50_Project_Project_C_Course\Project_C_Course.MCP
```
4.9.2 Sample 2 - Determining the Project Path

In this sample, the procedure for determining the project path of a WinCC project is outlined. The sample has been configured at the Button2 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button2

```c
#include "apie(sp.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty;
{
  BOOL bRet = FALSE;
  char szProjectFile[MAX_PATH+1];
  char* psz = NULL;
  CMN_ERROR Error;

  // get the project file *.ncp
  bRet = DMGetRuntimeProject(szProjectFile, MAX_PATH+1,&Error);

  // check return value
  if (bRet == FALSE)
    {
      printf("\r\n\nError in DMGetRuntimeProject()\r\n\n\n",Error.szErrorText);
      return;
    }

  // search for last backslash
  psz = strrchr(szProjectFile, '\');

  // cut string after last backslash
  if (psz != NULL)
    *(psz+1) = 0;

  // display project path
  printf("\nProjectPath:\n\n\n",szProjectFile);
}
```

- In the first section, a `bRet` variable of the `BOOL` data type is defined and initialized.
- Next, a `szProjectFile` string variable for accepting the project name is defined. In addition, a string variable is defined as `char*` and initialized with `NULL`.
- Next, a variable of the `CMN_ERROR` data type is defined.
- The project name is determined via the `DMGetRuntimeProject()` API function.
- Next, the return value of the `DMGetRuntimeProject()` API function is checked.
- Next, the `strrchr()` function searches for the last position of the `\` character in the determined name of the project file. One position after the found character, a 0 is inserted. Only the path specification of the project file - without the name of the project file itself - will remain.
- Next, the determined project path is output. This output is displayed in the next section.
Output in the Diagnostics Window

The sample described in this section generates the following output in the *diagnostics window*:

ProjectPath:
\\ZIP\WS5\WinCC50_Project_Project_C_Course\
4.9.3 Sample 3 - Determining the Project Path via a Project Function

In this sample, the determination of the project folder illustrated in the previous sample is shifted to a project function. The sample has been configured at the Button3 object displayed below at Event ➔ Mouse ➔ Mouse Action.

**Project Function GetProjectPath()**

```c
BOOL GetProjectPath(char* lpstrProjectPath)
{
    BOOL bRet = FALSE;
    char szProjectFile[_MAX_PATH+1];
    char* psz = NULL;
    COM_ERROR Error;
    bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);
    if (bRet == FALSE)
    {
        return FALSE;
    }
    psz = strchr(szProjectFile, '\\');
    if (psz == NULL)
    {
        return FALSE;
    }
    *(psz+1) = 0;
    strcpy(lpstrProjectPath, szProjectFile);
    return TRUE;
}
```

- A string variable is transferred to the project function, in which the determined project path is written. The caller of the function must ensure that enough memory space has been reserved for the string variable. If the function has been executed successfully can be seen by its return value.
- The procedure for determining the project path follows the same principle shown in the previous sample.
- The project path determined is copied to the transferred string variable via the *strcpy()* function.
C-Action at Button3

```c
#include "apdevasp.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    BOOL bRet = FALSE;
    char szProjectPath[_MAX_PATH+1];

    // project function to get project path
    bRet = GetProjectPath(szProjectPath);

    // check return value
    if (bRet == FALSE)
    {
        printf("\r\nError in GetProjectPath()\r\n");
        return;
    }

    // display project path
    printf("\r\nProjectPath: \r\n", szProjectPath);
}
```

- In the first section, a `bRet` variable of the `BOOL` data type is defined and initialized.
- Next, a `szProjectPath` string variable for accepting the project path is defined.
- Using the previously created `GetProjectPath()` project function, the project path is determined. Afterwards, the return value of the `project function` is checked.
- Next, the determined project path is output.
4.9.4 Sample 4 - Determining the Installation Folder

In this sample, the procedure for determining the installation folder of WinCC is outlined. The sample has been configured at the Button4 object displayed below at Event Mouse → Mouse Action.

C-Action at Button4

```c
#include "spdefap.h"

void OnClick(char* lpzPictureName, char* lpzObjectName, char* lpzProperty)
{
    char szProjectFile[_MAX_PATH+1];
    DM_DIRECTORY_INFO dmDirInfo;
    CMN_ERROR Error;
    char* psz = NULL;

    // get the project file name
    if ( DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error) == FALSE )
    {
        printf("\n\nAn Error in DMGetRuntimeProject()\n\n",Error.szErrorText);
        return;
    }

    // get wincc directories
    if ( DMGetProjectDirectory("",szProjectFile,&dmDirInfo,&Error) == FALSE )
    {
        printf("\n\nAn Error in DMGetProjectDirectory()\n\n",Error.szErrorText);
        return;
    }

    if ( (psz = strchr(dmDirInfo.szGlobalLibDir,'\')) != NULL )
    {
        *psz = 0;
    }
    if ( (psz = strchr(dmDirInfo.szGlobalLibDir,'\')) != NULL )
    {
        *(psz+1) = 0;
    }

    // display installation directory
    printf("\n\nInstallationDirectory:\n\n",dmDirInfo.szGlobalLibDir);
}
```

- In the first section, a `szProjectFile` string variable for accepting the name of the project file is defined.
- Next, a `dmDirInfo` variable for accepting the path information is defined. This is a variable of the `DM_DIRECTORY_INFO` structure type.
- Next, a variable of the `CMN_ERROR` data type is defined.
- Next, a `char*` string variable is defined and initialized with `NULL`.
- The project name is determined via the `DMGetRuntimeProject()` API function.
- Via the `DMGetProjectDirectory()` API function, the `dmDirInfo` variable is filled with the path information. One of the paths contained in the variable is the path to the Global Library folder. This path is stored in the `szGlobalLibDir` structure element. The Global Library folder is a subfolder of the WinCC installation folder.
• With the first strrchr() function, the last \ character is searched for in the determined path and replaced by a 0. With the second strrchr() function, the last \ character in the remaining path is searched for. One position after, a 0 is inserted.
• Next, the determined installation folder is output. This output is displayed in the next section.

Output in the Diagnostics Window

The sample described in this section generates the following output in the diagnostics window:

InstallationDirectory:
C:\Siemens\WinCC\
4.9.5 Sample 5 - Determining the Computer Name

In this sample, the procedure for determining the local computer name is outlined. The sample has been configured at the Button5 object displayed below at Event ➤ Mouse ➤ Mouse Action.

C-Action at Button5

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert;
{
    #pragma code (" Kernel32. DLL");
    BOOL GetComputerNameA(LPSTR ComputerName, LPDWORD pdwSize);
    #define MAX_COMPUTERNAME_LENGTH 15
    #pragma code();
    BOOL bRet = FALSE;
    char szComputerName[MAX_COMPUTERNAME_LENGTH + 1];
    DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;
    bRet = GetComputerNameA(szComputerName, &dwSize);

    //check return value
    if (bRet == FALSE)
    {
        printf("\n\nComputerName: \n\nUnknown Computer\n\n");
        return;
    }

    //display project file
    printf("\n\nComputerName:\n\n", szComputerName);
}
```

- In the first section, the Windows DLL Kernel32 is integrated. Since only one function of the DLL is required, this function is declared directly. In addition, a symbolic constant for the maximum length of the computer name is defined.
- Next, a `bRet` variable of the `BOOL` data type is defined and initialized.
- Next, a `szComputerName` string variable for accepting the computer name is defined. In addition, a variable of the `DWORD` data type is defined, which is initialized with the length of the previously defined string variable.
- The name of the local computer is determined by the `GetComputerNameA()` Windows function. This name is written to the `szComputerName` string variable transferred.
- Next, the return value of the `GetComputerNameA()` Windows function is checked.
- Next, the determined computer name is output.
4.9.6 Sample 6 - Determining the User Name

In this sample, the procedure for determining the user currently logged on to Windows NT is outlined. The sample has been configured at the Button6 object displayed below at Event → Mouse → Mouse Action.

![User Name Button]

C-Action at Button6

```c
#include "advapi32.dll"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    #pragma code ("advapi32.dll");
    #define UNLEN 256

    BOOL bRet = FALSE;
    char szUserName[UNLEN + 1];
    DWORD dwSize = UNLEN + 1;

    bRet = GetUserNameA(szUserName, &dwSize);

    // check return value
    if (bRet == FALSE)
    { 
        printf("\r\n\r\nUnknown User\r\n");
        return;
    }

    // display project file
    printf("\r\n\r\n\r\n", szUserName);
}
```

- In the first section, the Windows DLL advapi32 integrated. Since only one function of the DLL is required, this function is declared directly. In addition, a symbolic constant for the maximum length of the user name is defined.
- Next, a `bRet` variable of the `BOOL` data type is defined and initialized.
- Next, a `szUserName` string variable for accepting the user name is defined. In addition, a variable of the `DWORD` data type is defined, which is initialized with the length of the previously defined string variable.
- Via the `GetUserNameA()` Windows function, the name of the user currently logged on to Windows NT is determined. This name is written to the `szUserName` string variable transferred.
- Next, the return value of the `GetComputerNameA()` Windows function is checked.
- Next, the determined user name is output.
4.10 Windows API

The samples described in this chapter are configured in the cc_0_startpicture_00.PDL and cc_2_keyboard_01.PDL pictures of the WinCC project Project_C_Course.

Windows Application Program Interface

In addition to the WinCC API, the entire Windows API can also be utilized in a WinCC project. This provides almost unlimited access to the system. The following samples provide you with an overview of this topic. With the help of these samples, the general procedure for the application of the Windows API is shown. This is however, not an exhaustive treatment of the Windows API. The functions of the Windows API are also located in various DLLs, just like the functions of the WinCC API. The declarations of the functions are located in various header files. The integration of the DLLs follows the same principle that has been used for the integration of WinCC DLLs. The following sample program code illustrates the integration.

```c
#pragma code("comdlg32.dll")
#pragma include "comdlg32.h"
#pragma code()```

4.10.1 Sample 1 - Setting Windows Properties

This sample shows, how the properties of a Windows window can be changed. In this sample, the title and geometry of the runtime window is changed. The sample is configured at Event ➔ Miscellaneous ➔ Open Picture of the cc_0_startpicture_00.PDL start picture.

C-Action at the Start Picture

```
#include "apdsiap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpsz)
{
    // get handle of runtime window
    HWND hWnd = NULL;
    hWnd = FindWindow(NULL, "WinCC-Runtime - '");

    // get text of runtime window
    SetWindowText(hWnd, "WinCC C-Course");
    // set position and size of runtime window
    SetWindowPos(hWnd, HWND_TOP, 0, 0, 1024, 768, 0);

    // active the first chapter
    SetTagByte("NO6i_corg_bar_t", 0);

    // CreateExternalTags();
}
```

- The Windows functions used in this sample are already known to the WinCC project. Therefore no Windows DLL needs to be loaded.
- In the first section, a variable of the HWND type is defined and initialized with NULL. This variable is a so-called window handle - a pointer pointing to a Windows window.
- Via the FindWindow() Windows function, the window handle of a Windows window can be determined by specifying its window title. If the default title of the runtime window is indicated, its window handle can be determined.
- Via the SetWindowText() Windows function, the title of the runtime window can be changed. In this sample, the title is changed to WinCC C-Course.
- Via the SetWindowPos() Windows function, the position and dimensions of the runtime window displayed on the screen can be specified. In this sample, the runtime window is positioned at the top left corner (position 0/0) of the screen and is sized to 1024 by 768.
- The remaining statements of the above program code perform initializations that are not relevant for this sample.
4.10.2 Sample 2 - Reading the System Time

This sample shows how the system time can be read and displayed. In this sample, the time and date are displayed. The sample is configured at the \textit{cc_0_startpicture_00.PDL} picture.

\begin{verbatim}
#include "apifile.h"

char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    CAction code("kernel32.dll")
    VOID GetLocalTime(LPSYSTEMTIME lpSystemTime);
    SYSTEMTIME sysTime;
    char szTime[5] = "";
    GetLocalTime(&sysTime);
    sprintf(szTime, "%02d:%02d", sysTime.wHour, sysTime.wMinute);
    return szTime;
}
\end{verbatim}

- In the first section, the Windows DLL Kernel32 is integrated. Since only one function of the DLL is required, this function is declared directly.
- Next, the \texttt{sysTime} variable of the \texttt{SYSTEMTIME} data type is defined. This is a structure type, which stores the system time.
- Next, a \texttt{szTime} string variable for accepting the current time in the hh:mm format defined.
- Via the \texttt{GetLocalTime()} Windows function, the current system time is written to the \texttt{sysTime} variable.
- Next, the current system time is set to the hh:mm format and returned as a return value via the \texttt{sprintf()} function. At \textit{Property} \rightarrow \textit{Miscellaneous} \rightarrow \textit{Tooltip Text}, another \texttt{C-Action} following the steps described is created. This \texttt{C-Action} delivers the current date.
- The function is executed in 1s cycles.
4.10.3 Sample 3 - Playing Sound Files

This sample shows how sound files can be played. In this sample, a sound file is played if there is a switch from the navigation bar Basics to the navigation bar WinCC and Windows API and vice versa. The sample is configured in the cc_2_keyboard_01.PDL picture at the Button1 object.

![Diagram of navigation bar]

**Project Function CC_PlaySound()**

```c
#include "apdefap.h"

void CC_PlaySound(char* lpszSoundFile)
{
  #pragma code("winmm.dll")
  BOOL bRet = PlaySound(LPCSTR lpszSoundFile, HMODULE hModule, DWORD dwSound);
  #define SND_FILENAME 0x0020000L
  #define SND_ASYNC 0x0001
  #pragma code()

  bRet = FALSE;
  char szProjectPath[_MAX_PATH];
  char szSoundPath[_MAX_PATH];
  GetProjectPath(szProjectPath);
  sprintf(szSoundPath, "%sSound\%s", szProjectPath, lpszSoundFile);
  bRet = PlaySound(szSoundPath, NULL, SND_FILENAME|SND_ASYNC);
  if (bRet == FALSE)
    MessageLoop((WORD)-1);
}
```

- In the first section, the `apdefap.h` file is integrated. With this, other project functions can also be called from the present project function.
- The function header defines a string variable as the transfer parameter. With this variable, the name of the sound file to be played is transferred.
- In the next section, the Windows DLL `winmm` is integrated. Since only one function of the DLL is required, this function is declared directly. In addition, two symbolic constants are defined.
• The \textit{project function} assumes that a sound subfolder exists in the project folder. In this folder, the sound files used in the project are stored. The path to the desired sound file is composed of the project path, the name of the sound folder and the transferred name of the sound file. It will be stored in the \textit{szSoundPath} variable.

• The sound is played via the \texttt{PlaySound()} Windows function. If the sound file cannot be played, a brief beep sound is generated instead by the \texttt{MessageBeep()} Windows function.
4.10.4 Sample 4 - Starting a Program

This sample shows the procedure for starting a program. For this purpose, an already existing standard function is utilized, which uses the Windows API. The sample is configured at the cc_0_startpicture_00.PDL picture.

**Standard Function ProgramExecute()**

```plaintext
unsigned int ProgramExecute( char* Program_Name )
{
    // This function will start any Windows Program
    // if return value > 31 the program started successfully
    return ( WinExec( Program_Name, SW_SHOWNORMAL ) );
}
```

- The standard function ProgramExecute() simply forwards the parameter transferred to the WinExec() Windows function. The return value of the WinExec() function is forwarded to the caller of the ProgramExecute() function. The start of a program was successful, if the return value is greater than 31.

**C-Action at Graphic Object Execute**

```plaintext
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    ProgramExecute("calc.exe");
}
```

- Via the standard function ProgramExecute(), the program calc.exe is started. This is the Windows calculator program. No path needs to be specified, since this is not necessary for programs located in the Windows folder.
4.11 Standard Dialogs

In the WinCC project Project_C_Course, samples pertaining to the topic Standard Dialogs can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_12.PDL picture.

![Standard Dialogboxes](image)

General Information

The regular procedure for creating a dialog in WinCC consists of creating a WinCC picture and then have this picture displayed using a Picture Window. There is also the possibility to create standard dialogs in C-Actions or other functions. In this case, WinCC standard dialogs as well as Windows dialogs can be used.

In this chapter, the application of some of the available standard dialogs is shown. There a number of available standard dialogs that will not be mentioned here. Information about these dialogs can be found in the WinCC ODK and the Windows API documentation.
4.11.1 Sample 1 - Language Switch

This sample shows how the WinCC standard dialog for the language switch can be used. The sample has been configured at the Button1 object displayed below at Event \textit{Mouse \rightarrow Mouse Action.}

### C-Action at Button1

```c
#include "opcdlexp.h"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperties)
{

HWND hwndParent = NULL;
DWORD dwFlags = 0;
DWORD dwGetLocaleIDs[3] = { 0x0409, 0x0402, 0x040C },
UINT uSetIDArraySize = 3;
DWORD dwGetLocaleID;
BOOL bRet; 

CMN_ERROR Error;

hwndParent = FindWindow(NULL, "WinCC C-Course");

//set cs language (dWGetLocaleID contains selected language ID)

bRet = DMShowLanguageDialog(hwndParent, dwFlags, dwGetLocaleIDs,
    uSetIDArraySize, &dwGetLocaleID, &Error);

if (bRet == FALSE)
{
    printf("\nError in DMShowLanguageDialog()\n"
           "%s", Error.szErrorMessage);
    return;
}

//set rt language

bRet = SetLanguage(dwGetLocaleID);

if (bRet == FALSE)
{
    printf("\nError in SetLanguage()\n");
    return;
}
}
```

- In the first section, the variables used are defined. Among others, a vector consisting of the IDs of the three desired languages is defined.
- Via the \textit{FindWindow()} Windows function, the window handle of the runtime window is determined using its window title. Note that the window title specified in this code sample is not the default title of the runtime window.
- Via the \textit{DMShowLanguageDialog()} API function, the standard dialog for the language switch is displayed. A vector with the IDs of the languages to be displayed in the dialog is transferred to this function. The function writes the ID of the language selected by the user to the \textit{dwGetLocaleID} variable transferred.
• The return value of the \textit{DMShowLanguageDialog()} API function is checked. Among others, this return value can have the value \textit{FALSE}, if the user terminates the dialog by clicking on \textit{Cancel}.

• The dialog used here only switches the CS language. In order to switch the RT language, the \textit{internal function SetLanguage()} must be used. The ID of the language selected in the dialog is transferred to this function.

\textbf{Select Language Dialog}

If the \textit{C-Action} detailed above is executed, the following dialog will be displayed:
4.11.2 Sample 2 - Tag Selection

This sample shows, how the WinCC standard dialog for the tag selection can be used. The content of the tag selected from the dialog is displayed in an I/O Field. The sample has been configured at the Button2 object displayed below at Event $\rightarrow$ Mouse $\rightarrow$ Mouse Action.

![Tag Selection](image)

C-Action at Button2

```c
#include "trigger.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    // include file with the LinkType definitions
    #include "trigger.h"

    BOOL bSet;
    char szProjectFile[_MAX_PATH+1];
    CMN_ERROR Error;
    Hwnd hwndParent = NULL;
    DM_VARKEY dmVarKey;
    LINKINFO link;

    // select tag
    if ( DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error) == FALSE)
    {
        printf("\n\nError in DMGetRuntimeProject()\n\n" , Error.szErrorText);
        return;
    }
    hwndParent = FindWindow(NULL,"WinCC C-Course");

    if ( DMShowVarDatabase(szProjectFile,hwndParent,NULL,NULL, &dmVarKey,&Error) == FALSE)
    {
        printf("\n\nError in DMShowVarDatabase()\n\n" , Error.szErrorText);
        return;
    }

    // display tag selection
    SetText(lpszPictureName,"TagName",dmVarKey.szName);

    link.LinkType = CMN_BT_VARIABLE_DIRECT;
    link.dmCycle = 0;
    strcpy(link.szLinkName,dmVarKey.szName);
    PDLRTSetLink(0,lpszPictureName,"TagValue","OutputValue",
        &link,NULL,NULL,&Error);
}
```

- In the first section, the trigger.h file is integrated. This file contains the definition of a symbolic constant used in this sample.
- In the next section, the variables used are defined. Among others, a dmVarKey variable for accepting information about the WinCC tag selected in the dialog and a link variable for accepting information about the tag connection are defined.
- The project name is determined via the DMGetRuntimeProject() API function.
- Via the FindWindow() Windows function, the window handle of the runtime window is determined using its window title.
• Via the `DMShowVarDatabase()` API function, the tag selection dialog is opened. Information about the WinCC tag selected in the dialog is stored in the `dmVarName` variable transferred.

• If a tag has been selected, its name will be displayed in a `Static Text` field and its content in an `I/O Field`.

**Select Tag Dialog**

If the *C-Action* detailed above is executed, the following dialog will be displayed:
4.11.3 Sample 3 - Error Box

This sample shows, how a Windows error box can be displayed. The sample has been configured at the Button3 object displayed below at Event → Mouse → Mouse Action.

![Error Box]

C-Action at Button3

```c
#include 'acddatap.h'
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
    HWND hWnd = NULL;
    hWnd = FindWindow(NULL, "WinCC C-Course");
    MessageBox(hWnd, "WinCC C-Course raised unknown Exeption!!!", "Error", MB_OK | MB_ICONSTOP | MB_APPLMODAL);
}
```

- In the first section, a hWnd variable of the HWND data type is defined. Via the FindWindow() Windows function, the window handle of the runtime window is assigned to this variable.
- Via the MessageBox() Windows function, an error box is opened. As the second parameter, the error text and as the third parameter, the title of the error box are specified. The fourth parameter specifies the appearance and behavior of the error box. The error box only contains an OK button (MB_OK), displays the error symbol (MB_ICONSTOP) and is modal (MB_APPLMODAL). In this way, the user must acknowledge the error box first, before he or she can proceed.
- If a tag has been selected, its name will be displayed in a Static Text field and its content in an I/O Field.

Error Box

If the C-Action detailed above is executed, the following error box will be displayed:

![Error Box Image]
4.11.4 Sample 4 - Question Box

This sample shows, how a Windows question box can be displayed and how another action can be performed depending on the button pressed by the user. The sample has been configured at the Button4 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button4

```c
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    HWND hWnd = NULL;
    int iRet;
    hWnd = FindWindow(NULL,"WinCC C-Course");
    iRet = MessageBox(hWnd,"Do you want to do something?"."Question".
                   MB_YESNO|MB_ICONQUESTION|MB_APPLMODAL);
    printf("\n\nExample 3\n\n");
    if (iRet == IDYES)
    {
        printf("User selected YES button\n");
    }
    else if (iRet == IDNO)
    {
        printf("User selected NO button\n");
    }
}
```

- In the first section, a hWnd variable of the HWND data type is defined. In addition, a iRet variable of the int type is defined.
- Via the FindWindow() Windows function, the window handle of the runtime window is determined using its window title.
- Via the MessageBox() Windows function, a question box is opened. The fourth parameter specifies the appearance and behavior of the question box. The question box only contains a Yes and No button (MB_YESNO), displays a question mark (MB_ICONQUESTION) and is modal (MB_APPLMODAL). The return value of the function is stored in the iRet variable.
- In the last section, the return value of the function is analyzed. If the dialog was ended with Yes, the return value is IDYES, if the dialog was ended with No, the return value is IDNO.
Question Box

If the C-Action detailed above is executed, the following question box will be displayed:

![Question Box Image]
4.11.5 Sample 5 - Open Standard Dialog

This sample shows, how the standard dialog for opening a file can be displayed. The sample has been configured at the Button5 object displayed below at Event → Mouse Action.

C-Action at Button5

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

#pragma code ('comdlg32.dll')
#pragma include 'comdlg32.h'
#pragma code()’

BOOL bRet;
OPENFILENAME ofn;
char szFilter[] = "Textfiles (*.txt) | All Files all *.*"
char* psz;
char szFile[MAX_PATH+1];
char szInitialDir[MAX_PATH+1] = 'C:\';
ofn.nStructSize = sizeof(OPENFILENAME);
ofn.lpstrOwner = GetWindow(NULL, "WinCC C-Course");
for (psz = szFilter; *psz; psz++)
{
    if (*psz == '|')
    {
        *psz = 0;
    }
}
ofn.lpstrFilter = szFilter;
ofn.lpstrFile = szFile;
ofn.lpstrInitialDir = szInitialDir;
GetProjectPath(szInitialDir); // if function fails initial
directory is "C:\"
ofn.lpstrInitialDir = szInitialDir;
bRet = GetOpenFileName(&ofn);
if (bRet == FALSE)
{
    printf("Error in GetOpenFileName()\n");
    return;
}
printf("Selected File (Path+Name):\n\n", ofn.lpstrFile);
}
```

- In the first section, the Windows DLL `comdlg32` is integrated.
- In the next section, the variables required are defined. Among others, a `ofn` variable of the `OPENFILENAME` structure type is defined.
- In the next section, the `ofn` variable is filled with information.
- The `ofn` variable is transferred to the `GetOpenFileName()` Windows function. This function opens the file selection dialog. The name of the file selected by the user is stored in the `ofn` variable. The name of the file selected is output.
Open Standard Dialog

If the *C-Action* detailed above is executed, the following dialog will be displayed:

```
<table>
<thead>
<tr>
<th>Look in:</th>
<th>File name: Textdatei.txt</th>
<th>Files of type: Textfiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pde</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrtSync</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundancy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Additional Samples

The subsequent samples in this chapter deal - just like sample 5 - with the standard file dialogs.
In sample 6, the *Save As* dialog is used.
In sample 7, the *project function GetFileName()* is created, which facilitates dealing with the standard file dialogs. This function can, depending on the symbolic constant transferred, display an *Open* or a *Save As* dialog. Symbolic constants available are *GFN_OPEN* and *GFN_SAVE*. 
4.12 Files

In the WinCC project Project_C_Course, samples pertaining to the topic Files can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_13.PDL picture.

**Open File**

In C, a file is viewed - independent of its content - as a collection of characters. Before a file can be used in a C-Action or another function, it must be opened. If the work with a file is complete, the file should be closed.

A file is opened using the fopen() function. In the program code displayed below, the application of the fopen() function is shown.

```c
FILE* pFile = NULL;
pFile = fopen("C:\\Test.txt", 'r');
```

In order to be able to work with the file, a pointer pointing to it must be defined. For this purpose, the FILE* data type is available. The fopen() function returns a pointer pointing to the opened file or NULL, if opening the file has failed. As the first parameter, the name of the file to be opened with path specification must be transferred to the fopen() function. As the second parameter, the mode with which the file is opened (e.g. for reading) is transferred to the function. The values that can be specified for the mode are listed in the table below.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Opens a file for reading. The return value is NULL if the file does not exist or there is no read authorization.</td>
</tr>
<tr>
<td>w</td>
<td>Opens a file for writing. The return value is NULL if the file does not exist or there is no write authorization.</td>
</tr>
<tr>
<td>a</td>
<td>Opens a file for being attached to the end. If the file does not exist, it is created. The return value is NULL if no file can be created or the file cannot be written to.</td>
</tr>
<tr>
<td>r+</td>
<td>Opens a file for being read and written to alternately. The return value is NULL if the file does not exist or there are no read and write authorizations for the file.</td>
</tr>
<tr>
<td>w+</td>
<td>Creates a file for being read and written to alternately. If the file already exists, it will be deleted. The return value is NULL if no rights for these actions are available.</td>
</tr>
<tr>
<td>a+</td>
<td>Opens a file for being read or attached to the end. The file is created if it does not exist. The return value is NULL if no read and write rights for the file are available.</td>
</tr>
</tbody>
</table>
Close File

After completing the work with the file, the file should be closed. A file is closed using the fclose() function. In the program code displayed below, the application of the fclose() function is shown. The pointer pointing to the file to be closed is transferred to the function.

```c
fclose(pFile);
```

Writing To and Reading a File

For writing to a file, a function similar to the printf() function is available. This is the fprintf() function. The application of the fprintf() function follows the same principle as the application of the printf() function. The output, however, is not made to a Global Script Diagnostics Window, but to a file. As the first parameter, the function expects a pointer pointing to this file. In the program code displayed below, the application of the fprintf() function is shown.

```c
fprintf(pFile, "\%d\n\%f\n", iVar, dValue);
```

For reading a file, the fscanf() function is available. The fscanf() function is structured identically to the fprintf() function. However, instead of specifying the variables whose values are written to the file, the addresses of the variables are specified to which the contents of the file are written to.
4.12.1 Sample 1 - Securing Data

This sample shows how data can be written to a file. The data to be written is first read from WinCC tags. The sample has been configured at the Button1 object displayed below at Event → Mouse → Mouse Action.

C-Action at Button1

```c
#include "apidec.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
    FILE* pFile = NULL;
    char szFile[_MAX_PATH+10];
    int iData;
    float fData;

    //get project path
    if (GetProjectPath(szFile) == FALSE)
    {
        printf("\n\nError in GetProjectPath()\n\n");
        return;
    }

    //create file name
    strcat(szFile, "Data.txt");

    //open or create file to write
    pFile = fopen(szFile, "w+"):

    //check return value of fopen()
    if (pFile == NULL)
    {
        printf("\n\nError in fopen()\n\n");
        return;
    }

    //get data to write
    iData = GetTagSDWord("C321_course_file_i");
    fData = GetTagSFloat("C321_course_file_f");

    //write data
    fprintf(pFile, "%d\n%f\n", iData, fData);
    fclose(pFile);

    //output in diagnostics window
    printf("\n\nData written in file:\n\t\n\t\n\t\n\t\n\n", iData, fData);
}
```

- In the first section, the variables required are defined. Among others, a variable of the `FILE*` type is defined and initialized.
- Via the `project function GetProjectPath()`, the project path is determined.
- Next, the path to the file to be created is compiled by the `strcat()` function. This path is transferred to the `fopen()` function. With this function, the desired file is opened or created.
- In the next section, the data to be written is read from the WinCC tags.
- Via the `fprintf()` function, the data is written to the file. Afterwards, the file is closed again.
4.12.2 Sample 2 - Reading Data

This sample shows how data can be read from a file. The data read is written to WinCC tags. The sample has been configured at the Button2 object displayed below at Event ➔ Mouse ➔ Mouse Action.

C-Action at Button2

```c
#include "apndefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropName)
{
  FILE* pFile = NULL;
  char szFile[_MAX_PATH+10];
  int iData;
  float fData;

  // get project path
  if (GetProjectPath(szFile) == FALSE)
    {
    printf("\r\nError in GetProjectPath()\n\n");
    return;
  }

  // create file name
  strcat(szFile, "Data.txt");

  // open file to read
  pFile = fopen(szFile, "rt");

  // check return value of fopen()
  if (pFile == NULL)
    {
    printf("\r\nError in fopen()\n\n");
    return;
  }

  // read data
  fscanf(pFile,"%d\n%f\n", &iData, &fData);
  fclose(pFile);

  // set data
  GetTagDWord("S22i_course_file_1", &iData);
  SetTagFloat("F32i_course_file_1", fData);

  // output in diagnostics window
  printf("\r\nData read from file: Data.txt\n\r\n\r\n", iData, fData);
}
```

- In the first section, the variables required are defined. Among others, a variable of the FILE* type is defined and initialized.
- Via the project function GetProjectPath(), the project path is determined.
- Next, the path to the file to be opened is compiled by the strcat() function. This path is transferred to the fopen() function. With this function, the desired file to be read is opened.
- Via the fscanf() function, the data is read from the file. Afterwards, the file is closed again.
- In the next section, the data read is written to the WinCC tags.
4.12.3 Sample 3 - Reporting

This sample shows, how a report file can be created. A *project function* is created to which a report text is transferred. This text will then be written into the report file. The sample has been configured at the *Button3* object displayed below at *Event → Mouse → Mouse Action*.

---

**Project Function LogText()**

```c
#include "apdefsp.h"

BOOL LogText(char* lpszLogText)
{
    FILE* pFile = NULL;
    char szFile[_MAX_PATH+10];
    //get project path
    if (GetProjectPath(szFile) == FALSE)
    {
        printf("\n\nError in GetProjectPath()\n\n");
        return FALSE;
    }
    //create file name
    strcat(szFile, 'Log.txt');
    //open or create file to append
    pFile = fopen(szFile, "a+");
    //check return value of fopen()
    if (pFile == NULL)
    {
        printf("\n\nError in fopen()\n\n");
        return FALSE;
    }
    //append data
    fprintf(pFile, "%s - %s\n", GetLocalTimeString(), lpszLogText)
    fclose(pFile);
    return TRUE;
}
```

- A string variable is assigned to the function, which will be attached to the end of the report file.
- In the first section, the variables required are defined. Among others, a variable of the `FILE*` type is defined and initialized.
- Via the *project function* `GetProjectPath()`, the project path is determined.
- Next, the path to the file to be opened is compiled by the `strcat()` function. This path is transferred to the `fopen()` function. With this function, the desired file to be attached is opened.
- Via the `fprintf()` function, the report text transferred is entered into the file. Before each report entry, the current system time is entered via the *standard function GetLocalTimeString()*). Afterwards, the file is closed again.
C-Action at Button3

```c
#include "apdef.h"

void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProperty)
{
    if (LogText(GetTagChar("T081_course_file_1"))) == FALSE)
    {
        printf("\r\nError in LogText()\r\n");
    }
}
```

- In the C-Action, the content of a WinCC text tag is read and transferred to the previously created project function LogText(). Consequently, the content of the WinCC text tag is entered into the report file.
4.13 Dynamic Wizard

In the WinCC project Project_C_Course, samples pertaining to the topic Dynamic Wizard can be accessed by clicking on the navigation bar icon displayed below. The samples are configured in the cc_9_example_14.PDL picture.

General Information

The Dynamic Wizard is available in the Graphics Designer as an additional function. It can support the user with often repeated configuration processes. This reduces the configuration efforts and decreases possible configuration errors. The Dynamic Wizard is made up of various Dynamic Wizard functions. A large number of Dynamic Wizard functions are already available. They can be complemented by used-defined functions.

Working with the Dynamic Wizard

The Dynamic Wizard is displayed in the Graphics Designer via the View ➔ Toolbars ➔ Dynamic Wizard menus. The following shows the structure of the Dynamic Wizard.

In the various tabs, Dynamic Wizard functions already available are displayed. Via a ➔ on the desired Dynamic Wizard function, the function is started.

A Dynamic Wizard function consists of multiple pages that must be filled out by the user. These pages include a start page, a trigger page, various option pages and a final page, which summarizes the settings made in the previous pages.
4.13.1 Creation of Dynamic Wizard Functions

For the creation of user-defined Dynamic Wizard functions, a separate editor is available.
This editor is located in the \bin folder: is the dynwizedit.exe program.
Each Dynamic Wizard function is stored in a separate script file. For each of the languages
German, English and French, a separate script file exists. These script files - depending on
the language - are stored in the following folders:
WinCC InstallationFolder\Wscripts\Wscripts.deu
WinCC InstallationFolder\Wscripts\Wscripts.enu
WinCC InstallationFolder\Wscripts\Wscripts.fra

After starting the Dynamic Wizard Editor, select the desired language for which the new
Dynamic Wizard function is created from the toolbar.
A Dynamic Wizard function must follow a prescribed structure. In the scope of this manual,
two samples for the creation of Dynamic Wizard functions are provided. The script files, in
which these samples are located, are stored in the DynWiz subfolder - especially created for
this purpose - of the WinCC project Project_C_Course. These script files must be copied to
the folders listed above, where the default script files are located. Afterwards, the samples
can be opened from the Dynamic Wizard Editor.

Demo Wizard

In the Demo.wnf script file, a Dynamic Wizard named Demo Wizard has been created. This
Wizard displays basic functions which enable the user to enter data comfortably. However,
this Dynamic Wizard does not perform any actions.

Making a Motor Dynamic Wizard

In the Motor.wnf script file, a Dynamic Wizard named Making a Motor Dynamic has been
created. This Wizard has been especially created to add dynamics to a customized object
named Motor and cannot be used for any other object type. This customized object is stored
in the project library of the WinCC project Project_C_Course and can be inserted from
there into a picture. The customized object Motor is connected to a WinCC structure tag of
the Motor structure type via the Dynamic Wizard Making a Motor Dynamic. More
specifically, various C-Actions and tag connections are created at this object. It is assumed
that an internal WinCC text tag T08i_course_wiz_selected exists. With the help of this tag,
the currently selected motor object can be marked.

Compiling Script Files

A completely created Dynamic Wizard function must be compiled via the Dynamic Wizard
Compile Script menus and then be saved. In order to use the Dynamic Wizard
function in the Graphics Designer, it must be integrated into the database of the Dynamic
Wizard. This is done via the menus Dynamic Wizard Reading Wizard Script. The
script file to be read must be selected from the dialog displayed.
4.13.2 Structure of a Dynamic Wizard Function

The following explains the various sections that make up a Dynamic Wizard function.

Integration of Header Files and DLLs

The first part of a Dynamic Wizard function is the integration of the required header files. The most important file to be integrated is the dynamic.h file, in which the functions pertaining to the appearance of the Dynamic Wizard's user interface are declared. In addition, all desired DLLs of the Windows or WinCC API can be integrated here.

```c
#include "dynamic.h"
#pragma code("pdiwspapi.dll")
#include "pdiwapi.h"
#pragma code()
```

Language-Dependent Definitions

If the Dynamic Wizard function is to be available in several languages, a separate script file must be created for each language. Therefore, language-dependent texts should be defined before the program code. Then, a script file created for a certain language can simply be copied. Only the section with the language-dependent definitions must then be adapted.

```c
#include "defenu.h"
char* DynWizGroupName = "WinCC C-Course";
char* DynWizDynamicName = "Make a Motor Dynamic";
char* DynWizStartOption1 = "Select the desired Structure Tag!";
char* DynWizGenerateInfo = "The motor is made dynamic using the\r\n" "structure tag\r\n\r\n";
```

Properties List

There is the option to specify, if a Dynamic Wizard function can only be used for certain object types. This is done by specifying a list of object properties. If an object has one or more of the properties listed, the Dynamic Wizard function can be used for this object. This option has been applied to the Dynamic Wizard Making a Motor Dynamic to only make this Wizard functional for customized objects of the Motor type. This object type only has the properties Manual and Selection. If a blank properties list is used, a Dynamic Wizard function can be applied to all object types. In any case, a properties list must exist, even if it is blank.
**Processing Function**

The processing function is the function which performs the actual work of the *Dynamic Wizard* function after the *Finish* button is pressed. The name of this function must be specified in the system interface. An extensive processing function is presented in the *Dynamic Wizard Making a Motor Dynamic*.

**Info Function**

The info function summarizes the settings made by the user and displays them in summary form on the last page of the *Dynamic Wizard* function. The name of this function must be specified in the system interface.

```c
//this wizard can only be executed on the customized object "motor"
BEGIN_PROPERTY_SCHEME
  ( "Hand", VT_BOOL ),
  ( "Selection", VT_BOOL ),
END_PROPERTY_SCHEME
```

**System Interface**

Via the system interface, various properties of the new *Dynamic Wizard* function are specified. The meaning of the individual parameters are explained following the code sample.

```c
BEGIN_DYNAMICS
  {
    DynUIGroupName,  //group name
    DynUIDynamicName,  //dynamic name
    NULL,
    "logo16.bmp",  //use the default icon
    NULL,          //no help string is used
    "OnOption1",
    "OnOption2",
    NULL,
    "OnGenerate=",
    "OnShowGenerateInfo",
    JCR_TRIGGERS,
    { NULL, NULL },
  },
END_DYNAMICS
```

- The first parameter specifies in which tab the *Dynamic Wizard* function is displayed.
- The second parameter specifies under which name the *Dynamic Wizard* function is displayed.
- As the third parameter, *NULL* is always transferred.
- The fourth parameter contains the name of the icon used for the *Dynamic Wizard* function.
- As the fifth parameter, a help text describing the functionality of the *Dynamic Wizard* function in more detail can be transferred.
• As the sixth parameter, a list containing the names of the individual option pages of the function created is specified. This list must conclude with a NULL entry. A maximum of five option pages can be created.

• As the seventh parameter, the name of the processing function is specified, which is called after the Finish button is pressed.

• As the eighth parameter, the name of the function is specified, which summarizes the settings made in the option pages and displays them before the user presses the Finish button.

• As the ninth parameter, a list of triggers to be displayed on the trigger page is specified. For the most frequently occurring application cases, macros are available which will fill out this trigger list.

Global Variables

For each parameter to be set in the option pages, a global variable must be defined. This ensures that the parameters set are known to all functions created and can be worked with.

```c
//global vars
char g_MotorStructName[255] = "MotorStruct";
char* g_SelectedMotor = "TUS1_course_v12_selected";
DWORD dwTypes[1];
```

Option Pages

For each option page required, a separate function must be created. The names of these functions must be specified in the system interface.
5 Appendix

The appendix contains a collection of topics that have not been incorporated directly in the Configuration Manual.
5.1 Tips and Tricks

Additional samples for the configuration with WinCC.
5.1.1 Formatted Input/Output at the I/O Field

In order for an I/O field to display the value formatted or to transfer the input value formatted to the PLC, the following actions must be configured:
Action at the Output Value property of an I/O field (important: float, if decimal places are desired):

```c
Float a;
a=GetTagFloat("DB21_DW1");
return(a/100);
```

Action at the Input Value event of an I/O field (tag Var1 is an unsigned 16-Bit value):

```c
float a;
a=GetInputValueDouble(lpszPictureName,lpszObjectName);
SetTagFloat("Var1",a*100);
```
5.1.2 Object-Specific Actions at Open Picture

There are applications where actions at the property of one or more objects in a picture are to be performed only once at the opening of the picture. One option is to formulate a picture-specific action for the picture object at Events ➔ Miscellaneous ➔ Open Picture.

This, however, has the disadvantages that the action has to act on objects in the picture and thus the object names have to be specified fixed in the action. The objects can no longer be handled freely. This solution is not object-oriented.

There is an option to circumvent this problem:

- Define an internal tag (e.g. dummy) that is never updated or deliberately set. Set the trigger of the action at the object to upon change of this tag. At the opening of the picture in runtime, the action is activated once and would then only react again upon the change of the dummy tag, which cannot happen since this tag never changes.
5.1.3 WinCC Scope

General Information

WinCC Scope is a tool that supports you with the diagnosis of WinCC projects. It provides information about the activated project and the particular computer system. To work with Scope, a Web Browser such as the Internet Explorer is required. Additionally, the network protocol TCP/IP must be installed.

Start and Operation

If you have installed WinCC, then Scope has also been installed by default. Before Scope can be used, the WinCCDiagAgent.exe program must be started. It is located in the Siemens\WinCC\WinCCScope\bin folder. This program is a simple HTTP server. Following that, Scope can be activated via the Start menu. From the start page, a general description about the operation of WinCC Scope can be accessed via the How to use the new Diagnostics Interface link. Click on the link http://localhost to start Scope. From the list on the left, various information can be retrieved. In the System Info section, general information about the computer system is accessed, in the WinCC Info section, information about the currently active WinCC project is accessed.
5.1.4 Access to the Database

5.1.4.1 Access to the Database from MS Excel/MS Query

The following description for the access to the WinCC database refers to the application of Microsoft® Excel 97 with SR-1.

Access from MS Excel/MS Query

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Access from MS Excel/MS Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open MS Excel. Via the Data ➔ Get External Data ➔ Create New Query menus, the Choose Data Source dialog from MS Query is opened.</td>
</tr>
</tbody>
</table>

In the Databases tab, the New Data Source entry is selected. With the OK button, a new data source is created.
### Procedure: Access from MS Excel/MS Query

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>In the Create New Data Source dialog, the name of the new data source is specified. This name does not have to agree with the name of the WinCC database. As the driver, <strong>Sybase SQL Anywhere 5.0</strong> is selected. Clicking on the Connect... button opens the Connect to SQL Anywhere dialog, in which information required by the driver is entered. As the User ID, <em>dba</em> is entered and as the Password <em>sql</em>. Via the Browse button, the database to be edited is selected. Click on OK to accept the entries.</td>
</tr>
</tbody>
</table>

#### Connect to SQL Anywhere

- **User ID:** dba
- **Password:** ***
- **Data Source:** Example_01rt
- **Server Name:** <default>
- **Database:** Example_01rt
- **Database File:** D:\Handbuch\Example_01\EXAM
- **Start command:** C:\Siemens\Common\sqlany\dbms
- **Switches:**
  - Autostep Database
  - Microsoft Applications (Keys in SQLStatistics)
  - Prevent Driver not Capable errors
  - Delay AutoCommit until statement close
### Procedure: Access from MS Excel/MS Query

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Access from MS Excel/MS Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>If no data source has been configured for the selected database, the message <em>Data source name not found and no default driver specified</em> is displayed.</td>
</tr>
</tbody>
</table>

This message is acknowledged and the *Connect...* button pressed again. In the *Select Data Source* dialog, the *Machine Data Source* tab is selected. The CS as well as the runtime database of the currently running WinCC project are already available from the list of data sources. The names of these data sources start with *CC_* followed by the project name. The name of the data source representing the runtime database ends with the character *R*.

If, however, an arbitrary WinCC database is to be edited, a data source must first be created for it. This is done via the *New* button. On the first page of the *Create New Data Source Wizard* displayed, the *User Data Source* entry is selected. The page is completed by clicking on *Next*. On the next page, the driver *Sybase SQL Anywhere 5.0* is selected. The page is completed by clicking on *Next*. The last page of the Wizard is completed by clicking on *Finish*.

The *SQL Anywhere ODBC Configuration* dialog is opened, in which information required by the driver is entered. As the *User ID*, *dba* is entered, again, and as the *Password sql*. Via the *Browse* button, the database to be edited is selected.

The dialog is closed with *OK*.
### Access from MS Excel/MS Query

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Access from MS Excel/MS Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Close the <strong>Create New Data Source</strong> dialog with <strong>OK</strong>.</td>
</tr>
<tr>
<td></td>
<td>In the <strong>Select Data Source</strong> dialog, the newly created data source is selected and the dialog is closed with <strong>OK</strong>.</td>
</tr>
<tr>
<td></td>
<td>On the first page of the <strong>Query Wizard</strong> displayed, all available tables and columns are listed. The desired tables and columns are selected and the page is closed by clicking <strong>Next</strong>. On the next pages, filters for the data and their sort order can be set. On the last page, it is specified whether the data is processed further in MS Excel or in MS Query. The dialog is closed by clicking <strong>Finish</strong>.</td>
</tr>
<tr>
<td>5</td>
<td>In the <strong>Returning External Data to Microsoft Excel</strong> dialog displayed, the location of the tables to be inserted is specified. Additionally, the properties of the external data range can be specified. Close the dialog by clicking on <strong>OK</strong>.</td>
</tr>
</tbody>
</table>

### Access to the Database from MS Access

The following description for the access to the WinCC database refers to the application of Microsoft® Access 97 with SR-1.

#### Access via MS Access

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Access via MS Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open an Access database or create a new one. Via the <strong>File</strong> ➔ <strong>Get External Data</strong> ➔ <strong>Import...</strong> menus, the <strong>Import</strong> dialog is opened. As the <strong>File Type</strong>, the list entry <strong>ODBC Databases()</strong> is selected. This automatically open the <strong>Select Data Source</strong> dialog. In the <strong>Machine Data Source</strong> tab, a data source is selected. The CS as well as the runtime database of the currently running WinCC project are already available from the list of data sources. The names of these data sources start with <strong>CC_</strong> followed by the project name. The name of the data source representing the runtime database ends with the character <strong>R</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>If the desired WinCC database is not included in the list, it must be created first as a new data source by clicking on the <strong>New</strong> button. On the first page of the <strong>Create New Data Source</strong> Wizard displayed, the <strong>User Data Source</strong> entry is selected. The page is completed by clicking on <strong>Next</strong>. On the next page, the driver <strong>Sybase SQL Anywhere 5.0</strong> is selected. The page is completed by clicking on <strong>Next</strong>. The last page of the Wizard is completed by clicking on <strong>Finish</strong>.</td>
</tr>
</tbody>
</table>
### Step Procedure: Access via MS Access

The SQL Anywhere ODBC Configuration dialog is opened, in which information required by the driver is entered. As the User ID, *dba* is entered and as the Password *sql*. Via the Browse button, the database to be edited is selected. The dialog is closed with OK.

The newly created data source is selected in the Select Data Source dialog and the dialog is closed with OK.

3 In the following Import Objects dialog displayed, the desired database tables can be selected. By clicking on OK, they will be inserted into the Access database.

#### 5.1.4.3 Access to the Database from ISQL

Direct access to the WinCC database is possible using ISQL. This, however, is performed at your own risk, since the configuration data may become inconsistent as a result of editing or deleting tables.
Access via ISQL

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Access via ISQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start ISQL.EXE from the Siemens\Common\sqlany folder. The Interactive SQL Logon dialog is displayed. As the User ID, dba is entered and as the Password, sql. If you acknowledge with OK, the program is automatically connected with the currently opened WinCC database - namely the CS database. If, however, another database is to be accessed, for example the runtime database, this is done via the Command Connect menus. In the following dialog displayed, the same entries for the User ID and the Password are made. As the Database File, the desired database including its complete path is specified.</td>
</tr>
</tbody>
</table>
| 2    | In the Command window, SQL statements can now be entered which are carried out by clicking on the Execute button. Below are some examples of SQL statements:  
  - select* from systable: displays all table names  
  - select * from >: displays the content of the table with the name >  
  - unload table > to >: exports the table with the name > to the file with the name >  
  - drop table >: deletes the table with the name > |

5.1.4.4 Access to the Database from WinCC Scope

Access via WinCC Scope

<table>
<thead>
<tr>
<th>Step</th>
<th>Procedure: Access via WinCC Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Before starting WinCC Scope from the Windows Start menu, the application WinCCDiagAgent.exe located in the Siemens\WinCC\WinCCScope\bin folder must be activated.</td>
</tr>
<tr>
<td>2</td>
<td>On the first page, a general description about the operation of WinCC Scope can be accessed via the How to use the new Diagnostics Interface link. Click on the link <a href="http://localhost">http://localhost</a> to start Scope.</td>
</tr>
</tbody>
</table>
| 3    | On the left side, various functions can be selected from the list.  
  - Via the Database entry, general information about the WinCC database is displayed.  
  - Via the Database Query entry, individual tables of a database can be displayed. The CS database of the currently opened WinCC project is preset as the Data Source. The name of this data source starts with CC_ followed by the project name. The name of the data source representing the runtime database ends with the character R. Other data sources can also be displayed.  
  - Via the SQL Query entry, SQL statements can be applied to a selected data source. However, it is recommended to edit the WinCC database with SQL statements only if you have extensive knowledge about the system. Examples of SQL statements can be found in the previous section Access to the Database from ISQL. |
5.1.4.5 Export from the Database via C-Actions

The data export can also be activated from a WinCC runtime picture. For this purpose, it is possible to start an interactive SQL with a command line via ProgramExecute. The action to be executed is stored in a command file (in this sample: archive.sql).

C-Action, for Example at a Button

```c
#include "apdefep.h"
void OnClick(char* lpzPictureName, char* lpzObjectName, char* lpzPropertyName) {

    char* path = "C:\SIEMENS\Common\SQLANY\ISQL-q-b-c";
    char* parameters = "UID=DBA;PWD=SQL;DBN=CC_Project_37-10-21_03:53:27R";
    char* action = "read D:\WinCC\Project\archive.sql";

    char ExportArchive[200];

    sprintf(ExportArchive,"%s %s %s",path,parameters,action);

    ProgramExecute(ExportArchive);
}
```

- The `path` variable contains the path to the ISQL exe program with its call parameters.
- The `parameters` variable contains the entries for the database connection made in the Interactive SQL Logon dialog. These are:
  - UID (User ID): DBA
  - PWD (Password): SQL
  - DBN (Database Name): Name of the ODBC data source. The name of this data source starts with `CC_` followed by the project name and the date/time of the project creation. The name of the data source representing the runtime database ends with the character `R`. This name can be determined while the project is active from the Windows Control Panel ➔ ODBC ➔ User DSN Tab.
- The `action` variable indicates that the SQL statements listed in the `archive.sql` file are to be executed.
- The statements are summarized in `ExportArchives` and carried out with the `ProgramExecute()` function.

Note:
If an export is to be performed from a database other than the two project databases, the DBF parameter (the database file including the path to the database) should be specified instead of the DBN parameter. However, this approach does not work for the currently activated project database.
File Contents: archive.sql

```sql
select * from PDE#HD#ProcessValueArchive#Analog;
output to D:\WinCC\Projekt\archiv.txt; format ascii
```

In the opened database, the measured value archive `pde#hd#ProcessValueArchive#Analog` is selected and exported to the ASCII file `archive.txt` using the Output command.

### 5.1.4.6 Database Selections

The previously described `select` command in the command file selects tables. Subsets of these tables can be selected with additional parameters and then be exported using the `output` command. Below are some examples pertaining to this topic.

#### Selection of a Time Range

```sql
select * from PDE#HD#ProcessValueArchive#Analog where T between '1996-5-1 10:10:0.00' and '1996-6-1 10:10:0.00'
```

#### Selection starting from a Time Stamp

```sql
select * from PDE#HD#ProcessValueArchive#Analog where T > '1996-5-1 10:10:0.00'
```

#### Selection of a Process Value with and without Sorting

```sql
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
order by T
```

```sql
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
```

#### Selection of a Process Value using the Columns T (Time) and V (Value)

```sql
select T,V from PDE#HD#ProcessValueArchive#Analog where V > 100
order by T
```
5.1.5 Serial Connection

The following settings must be made to establish a serial connection:

**CP525 Settings:**

- **Message:** Parameter CP525 Name: P3964R
- **Procedure:** Component: RK Version: 01
- **Baud Rate:** 9600
- **Number of Stop Bits:** 1
- **Parity:** Even

In the PLC, SYNCHRONOUS is required in the startup circuit for the CP525 and SEND/RECEIVE ALL in the cyclic program.

**WinCC Settings:**

For optimization purposes, one of the two partners should have the priority *high*, preferably WinCC.
### 5.1.6 Color Table

The color values are composed from a large palette. The sixteen basic colors are:

<table>
<thead>
<tr>
<th>Color</th>
<th>Color Value (Hex)</th>
<th>Symbolic Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0x000000FF</td>
<td>CO_RED</td>
</tr>
<tr>
<td>Dark Red</td>
<td>0x00000080</td>
<td>CO_DKRED</td>
</tr>
<tr>
<td>Green</td>
<td>0x0000FF00</td>
<td>CO_GREEN</td>
</tr>
<tr>
<td>Dark Green</td>
<td>0x00000800</td>
<td>CO_DKGREEN</td>
</tr>
<tr>
<td>Blue</td>
<td>0x00FF0000</td>
<td>CO_BLUE</td>
</tr>
<tr>
<td>Dark Blue</td>
<td>0x00800000</td>
<td>CO_DKBLUE</td>
</tr>
<tr>
<td>Cyan</td>
<td>0x00FFFF00</td>
<td>CO_CYAN</td>
</tr>
<tr>
<td>Dark Cyan</td>
<td>0x00808000</td>
<td>CO_DKCYAN</td>
</tr>
<tr>
<td>Yellow</td>
<td>0x0000FFFF</td>
<td>CO_YELLOW</td>
</tr>
<tr>
<td>Dark Yellow</td>
<td>0x00000800</td>
<td>CO_DKYELLOW</td>
</tr>
<tr>
<td>Magenta</td>
<td>0x00FF00FF</td>
<td>CO_MAGENTA</td>
</tr>
<tr>
<td>Dark Magenta</td>
<td>0x00800080</td>
<td>CO_DKMAGENTA</td>
</tr>
<tr>
<td>Light Gray</td>
<td>0x00C0C0C0</td>
<td>CO_LTGRAY</td>
</tr>
<tr>
<td>Gray</td>
<td>0x00808080</td>
<td>CO_DKGRAY</td>
</tr>
<tr>
<td>Black</td>
<td>0x00000000</td>
<td>CO_BLACK</td>
</tr>
<tr>
<td>White</td>
<td>0x00FFFFFF</td>
<td>CO_WHITE</td>
</tr>
</tbody>
</table>

Symbolic constants are predefined externally by `#define`. Mixed colors are produced by using intermediate values from the palette. If color changes are created with the help of the Dynamic Dialog and the configured data is subsequently processed with C-Actions, the color values can be read as well, although they are in the decimal format.
5.2 Documentation of the S5 Alarm System

Task and Function of the S5 Alarm System

This document describes the functions and properties of the SIMATIC S5 software: S5 Alarm System.

The software is used to ensure the sequenced acquisition of binary messages and to process and buffer them. The program package provides the required software functionality within SIMATIC S5 to implement the sequenced message acquisition function of the WinCC system.

The principal operation of the software can be illustrated as follows: The software monitors the binary signal status of the messages that the user makes available in a message interface to the S5 alarm system. If a signal condition changes, the message is identified by its message number and date/time stamped. A 32-Bit process tag and an alphanumeric job/batch identifier are added to this data (if so configured by the user). The message block configured in this manner is, if required, buffered in a FIFO buffer. Buffering message data becomes necessary whenever more messages are issued per time unit than can be transferred to the WinCC system using the present bus connection. This functionality achieves a separation with respect to time between the sequenced message acquisition in the SIMATIC S5 and the higher-level WinCC alarm system and makes real-time capable message processing possible.

The message blocks generated by the S5 alarm system are made available to the S5 application program in a data block interface. By using S5 communication software, which has to be implemented by the user, this data is transferred via a bus connection (e.g. SINEC H1) to the higher-level WinCC alarm system. In WinCC, comprehensive message processing functions, such as visualization, archiving, reporting, etc., are available. The configuration of the S5 alarm system is carried out by the user through a data block interface (System DB 80). At this point, the user determines the system requirements, within which the alarm system is working. Specified here are the memory areas used by the S5 alarm system, the type and scope of the messages to be processed and the allocation of the assigned address areas.

This section describes the application and handling of the S5 alarm system in the SIMATIC S5 environment. An overview of the function and data blocks used by the software as well as the storage space required is provided. This is followed by an in-depth interface description of all the existing data interfaces between the S5 alarm system and the S5 application program. To facilitate the start with the S5 alarm system, a configuration sample is included.
5.2.1 Listing of the Software Blocks

The SIMATIC S5 software is located on the CD-ROM together with the samples pertaining to this manual and is stored under the `WINCC1ST.S5D` file name. The file contains the following function and data blocks for the S5 alarm system:

<table>
<thead>
<tr>
<th>FB</th>
<th>Name</th>
<th>Size</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB 80</td>
<td>SYSTEMFB</td>
<td>1114</td>
<td>Sequenced Reporting</td>
</tr>
<tr>
<td>FB 81</td>
<td>STARTUPFB</td>
<td>135</td>
<td>Startup and Initialization of the Sequenced Reporting</td>
</tr>
<tr>
<td>FB 82</td>
<td>PCHECK</td>
<td>574</td>
<td>Called by FB 81</td>
</tr>
<tr>
<td>FB 83</td>
<td>MBLOCK</td>
<td>699</td>
<td>Called by FB 80</td>
</tr>
<tr>
<td>FB 84</td>
<td>WRITE</td>
<td>94</td>
<td>Called by FB 80</td>
</tr>
<tr>
<td>FB 87</td>
<td>FULL</td>
<td>87</td>
<td>Called by FB 80</td>
</tr>
<tr>
<td>DB 80</td>
<td>System DB</td>
<td>512</td>
<td>Assigns Parameters to Alarm Logging</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td>2703</td>
<td></td>
</tr>
</tbody>
</table>

Table 1

The minimum storage space requirement depends on the configuration of the S5 alarm system. The following data blocks are always required in addition.

- **FIFO Buffer (min.)**: 2 DB = 1024 Bytes
- **Transfer Mailbox to WinCC**: 1 DB = 512 Bytes

For each offset or parameter data block, an additional 512 Bytes must be included. The exact calculation of the size of the offset and parameter data blocks is shown in the chapters Structure of the Offset Data Block and Structure of the Parameter Data Block.
5.2.2 Hardware Requirements

The function blocks specified for the S5 alarm system in table 1 require the following hardware for the proper execution:

<table>
<thead>
<tr>
<th>PLC</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLC 115U</td>
<td>CPU 944 *, CPU 945</td>
</tr>
<tr>
<td>PLC 135U</td>
<td>CPU 928B</td>
</tr>
<tr>
<td>PLC 155U</td>
<td>CPU 946/ 947, CPU 948</td>
</tr>
</tbody>
</table>

Table 2

* Only the CPU 944 with two PG interfaces has a system clock. These CPUs have an internal clock which allows them to provide the current date/time for the generation of the message blocks.

For every WinCC channel set up, a current date/time message is cyclically written to the SIMATIC S5 CPU. The internal clock of the SIMATIC S5 is synchronized with the system clock of WinCC through the function block **FB 86 : MESS:CLOCK**.
5.2.3 Integration of the S5 Alarm System into the SIMATIC S5 Application Program

To integrate the SIMATIC S5 software for the alarm system into the SIMATIC S5 application program, the following steps must be performed:

All blocks specified in Table 1 must be transferred from the WinCCST.S5D file to the corresponding PLC.

If not already implemented by default or if they are not available in the PLC, transfer the data handling blocks for the PLC concerned.
Step | Procedure: Integrating the Alarm System
---|---
1 | All blocks specified in Table 1 must be transferred from the WinCCST.S5D file to the corresponding PLC.
2 | If not already implemented by default or if they are not available in the PLC, transfer the data handling blocks for the PLC concerned.
3 | Assign the parameters to the data block DB 80 according to the chapter Assigning Parameters to the DB80.
4 | Setup of the data blocks for the send mailbox, the FIFO buffer, the message offset and, if required, the message parameter according to the chapter Setup of the Data Blocks.
5 | Initialization of the offset data blocks for the various message categories according to the chapter Initialization of the Offset Data Blocks.
6 | Specification of process tags, job and batch identifier for the individual messages in the application program.
7 | Call of the following blocks in the startup OBs (OB 20, OB 21, OB 22):
   - SPA HTB : SYNCHRON (the data handling block of the CPU concerned)
   - SPA FB 81 : STARTUPFB
8 | Call the following blocks in the OB 1:
   - for the cyclic processing of messages SPA FB 80 : SYSTEMFB
   - a function block created by the user for transferring the message blocks to the higher-level WinCC system
9 | Additional functions are added according to the following chapters:
   - The synchronization of date and time through FB 86 : MESS:CLOCK.

Table 3

![Diagram](image)

Figure 1
General Description of the S5 Alarm System
The following components of the S5 alarm system will be described:

- Offset data block
- Parameter data block
- Message block
- FIFO buffer
- Send mailbox
- System data block

Relationship among the different components:

![Diagram showing the relationship among the different components.]

Figure 2

Before the message acquisition system can monitor and acquire messages, the messages have to be configured in the corresponding data blocks. There are four different message categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition: Message Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Message without parameters</td>
</tr>
<tr>
<td>2</td>
<td>Message with process tags (2 DWs)</td>
</tr>
<tr>
<td>3</td>
<td>Message with process tags (2 DWs) and job/batch identifier (3 DWs)</td>
</tr>
<tr>
<td>4</td>
<td>Message with process tags (2 DWs), job/batch identifier (3 DWs) and reserve (3 DWs)</td>
</tr>
</tbody>
</table>

Table 4

For the alarm system, a date/time stamp can be specified globally for the message block generation. If the date/time stamp is missing, the corresponding information is added to the message blocks by the WinCC system.
5.2.3.1 Structure of the Offset Data Block

The offset data block has the same structure for all four message categories. The corresponding data block address is specified for every required message category in the system data block DB 80.

Offset data block for the corresponding message category:

<table>
<thead>
<tr>
<th>DW</th>
<th>Content</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW 0</td>
<td>Not assigned</td>
<td>Header</td>
</tr>
<tr>
<td>DW 1</td>
<td>Basic Message Number</td>
<td></td>
</tr>
<tr>
<td>DW 2</td>
<td>Address of the last Signal Status Block</td>
<td></td>
</tr>
<tr>
<td>DW 3</td>
<td>Not assigned</td>
<td></td>
</tr>
<tr>
<td>DW 4</td>
<td>Signal States of the Messages - Bit No.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>Signal Status Block 1</td>
</tr>
<tr>
<td>DW 5</td>
<td>Idle Status Bits</td>
<td></td>
</tr>
<tr>
<td>DW 6</td>
<td>Acknowledgment Bits</td>
<td></td>
</tr>
<tr>
<td>DW 7</td>
<td>Edge Trigger Flags</td>
<td></td>
</tr>
<tr>
<td>DW 8</td>
<td>Signal States of the Messages - Bit No.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>Signal Status Block 2</td>
</tr>
<tr>
<td>DW 9</td>
<td>Idle Status Bits</td>
<td></td>
</tr>
<tr>
<td>DW 10</td>
<td>Acknowledgment Bits</td>
<td></td>
</tr>
<tr>
<td>DW 11</td>
<td>Edge Trigger Flags</td>
<td></td>
</tr>
<tr>
<td>DW 12</td>
<td>Signal States of the Messages - Bit No.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>Signal Status Block 3</td>
</tr>
<tr>
<td>DW 13</td>
<td>Idle Status Bits</td>
<td></td>
</tr>
</tbody>
</table>

Table 5

The following elements will be described:

- Basic Message Number
- Offset Message Number
- Signal Status Block
- Address of the last Signal Status Block
- Signal States
- Idle Status Bits
- Acknowledgment Bits
- Edge Trigger Flags
5.2.3.2 Basic Message Number

Every message is assigned a certain message number by which it can be recognized from the messages issued. The message number consists of the basic message number and an offset message number.

A different basic message number has to be specified for every message category used. Continuously from this basic message number, messages of this category are differentiated through the offset message number.

The basic message number for the corresponding message category is specified in the DW 1 of the associated offset data block.

Special Case

If the message category 1 is being used, it is possible to use two offset data blocks. To achieve a continuous message numbering for this message category, the basic message number of the second offset data block has to be entered as the basic message number of the first offset data block plus its message capacity (1008 messages).

Calculating the message number:

\[ \text{Message Number} = \text{Basic Message Number} + \text{Offset Message Number} \]

Example:

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given:</td>
<td>Message category 1, continuous message numbering starting at: message number 10000</td>
</tr>
<tr>
<td>Required:</td>
<td>Basic message number of the two offset data blocks</td>
</tr>
<tr>
<td>10000</td>
<td>Basic message number of the first offset data block</td>
</tr>
<tr>
<td>10000 + 1008 = 11008</td>
<td>Basic message number of the second offset data block</td>
</tr>
</tbody>
</table>
5.2.3.3 Offset Message Number/Signal States of the Messages

The signal states of the messages are contained in the offset data blocks of the corresponding message category at the respective bit position of the offset message number. The offset message number of the corresponding message arises starting with the 16 Bits (Bits 0-15) of DW 4. The continuous numbering is performed in increments of four (DW8, DW12, etc.).

<table>
<thead>
<tr>
<th>Signal Status Block</th>
<th>Signal Status Block starts at Data Word</th>
<th>Bit Number 0 - 15 corresponds to the Offset Message Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0 - 15</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16 - 31</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>32 - 47</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>48 - 63</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>62</td>
<td>248</td>
<td>976 - 991</td>
</tr>
<tr>
<td>63</td>
<td>252</td>
<td>992 - 1007</td>
</tr>
</tbody>
</table>

Table 6

Calculating the offset message number:

\[
\text{Offset Message Number} = \text{Message Number} - \text{Basic Message Number}
\]

\[
\text{Offset Message Number} = (\text{Data Word} / 4 - 1) \times 16 + \text{Bit No. (0-15)}
\]

\[
\text{Offset Message Number} = (\text{Signal Status Block} - 1) \times 16 + \text{Bit No. (0-15)}
\]

Calculating the DB, DW, Bit No. from the offset message number:

\[
\text{Data Block} = \text{Offset Data Block}
\]

\[
\text{Data Word} = (\text{Offset Message Number} / 16 + 1) \times 4
\]

\[
\text{Bit No.} = \text{Offset Message Number} \mod 16
\]

In the case of message category 1, the Length of a data word may be greater than 252. Then the following applies:

\[
\text{Data Block} = \text{Offset Data Block} + 1
\]

\[
\text{Data Word} = \text{Data Word} - 252
\]

\[
\text{Bit No.} = \text{Bit No.}
\]
Example 1:

Given: DW 248, Bit 7, Basic Message Number = 10000
Required: Message Number

Signal Status Block = 248 / 4
= 62

Offset Message Number = (Signal Status Block - 1) * 16 + Bit No.
= (62 - 1) * 16 + 7 = 983

Message Number = Basic Message Number + Offset Message Number
= 10000 + 983 = 10983

The required message number is 10983.

Example 2:

Given: Message Category 1 with two Offset Data Blocks, Message Number = 12000, Basic Message Number = 10000

Required: DB, DW, Bit No.

Offset Message Number = Message Number - Basic Message Number
= 12000 - 10000 = 2000

Bit No. = Offset Message Number % 16 = 0

Data Word = (Offset Message Number / 16 + 1) * 4
= (2000 / 16 + 1) * 4 = 504

The data word is greater than 252.

Data Block = Offset Data Block + 1
Data Word = 504 - 252 = 252
Bit No. = 0

The message number 12000 can be found in the second offset data block of Message Category 1, Data Word 252, Bit No. 0.

5.2.3.4 Signal Status Block

The first signal status block starts at the data word address 4, the next signal status blocks follow in intervals of 4 data words (DW 8, DW 12, etc.).
See also Table 5 or Table 6.
For each offset data block, 63 signal status blocks are possible (signal status blocks 1 to 63).
A signal status block contains 16 signal states. This results in 63 * 16 = 1008 possible messages in an offset data block.
Structure of the Signal Status Block:

<table>
<thead>
<tr>
<th>DW</th>
<th>Bit Number</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Signal States</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Idle States</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Acknowledgment Bits</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Edge Trigger Flags</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7

Additional information about these four bit states can be found in this chapter.

Calculating the corresponding signal status block:

\[
\text{Signal Status Block} = \frac{\text{Offset Message Number}}{16} + 1
\]

\[
\text{Signal Status Block} = \frac{\text{Data Word}}{4}
\]

Calculating the data word with which the corresponding signal status block starts:

\[
\text{First Data Word of the Signal Status Block} = \text{Signal Status Block} \times 4
\]

5.2.3.5 Address of the last Signal Status Block

By specifying the DW address of the last signal status block occupied with messages, the number of possible messages of the corresponding message category is specified.

Calculating the last signal status block:

\[
\text{Last Signal Status Block} = \frac{\text{Required Messages of this Message Category}}{16}
\]

```
// Incompletely filled [16 messages] signal condition block
if (((\text{required messages of this message category}) \mod 16) == 0)
{
    \text{++ last signal message block;}
}
```

With message category 1, a message volume of more than 1008 messages may occur, in which case the following applies:

1st Offset DB:

\[
\text{Last Signal Status Block} = 63
\]

\[
\text{Address of the last Signal Status Block} = 63 \times 4 = 252
\]
2nd Offset DB:

Last Signal Status Block = (Required Messages of this Message Category - 1008) / 16

Calculating the DW address of the last signal status block:

\[
DW \text{ Address of the last Signal Status Block} = Last \text{ Signal Status Block} \times 4
\]

Example:

Given: 1030 Messages of Message Category 1

1st Offset DB:

Address of the last Signal Status Block

2nd Offset DB:

\[
\begin{align*}
\text{Required Messages - 1008} & \quad = \quad 1030 \ - \ 1008 = 22 \\
(\text{Required Messages - 1008}) / 16 & \quad = \quad 22 / 16 = 1 \\
(\text{Required Messages - 1008}) \% 16 & \quad = \quad 22 \% 16 = 6 \\
\text{Last Signal Status Block} & \quad = \quad 2 \\
\text{Address of the last Signal Status Block} & \quad = \quad 4 \times 2 = 8
\end{align*}
\]

5.2.3.6 Signal States

Position: 1st data word of the signal status block (refer to Table 5). The user must ensure that the signal states of the corresponding messages are entered in the data words provided by the offset data blocks of the corresponding message category. This can be performed by the control program through continuous process-accompanying signal updates.

5.2.3.7 Idle States

Position: 2nd data word of the signal status block (refer to Table 5). Under idle status of a signal we mean the signal level during the passive operating status. It defines whether a signal (message) is active low or high. This information is required to find out whether a message is coming in or going out. If a change of event has the negated status with regard to the idle status, then a message is coming in. In the case of a message going out, the status of the change of event is identical to that of the associated idle status. The idle states of the messages must be specified by the user at the corresponding positions.
5.2.3.8 Acknowledgment Bits

Position: 3rd data word of the signal status block (refer to Table 5).
Acknowledgment bits are not configured but evaluated within the running program. Here, the messages are acknowledged directly by the higher-level PC in accordance with the corresponding acknowledgment philosophy. These message-related acknowledgments are sent by the PC to the corresponding PLC together with the configured messages of the integrated message acquisition system.
The corresponding acknowledgment bit is set one PLC cycle long by the S5 alarm system. The application program has to evaluate this information accordingly.

5.2.3.9 Edge Trigger Flags

Position: 4th data word of the signal status block (refer to Table 5).
The edge-triggered flags are used to determine change of events that may have occurred (change of message). They are not configured but evaluated within the S5 alarm system.

5.2.3.10 Structure of the Parameter Data Block

For message categories 2 to 4, it is necessary to configure parameter data blocks for additional data of the message concerned, in addition to an offset data block. The signal status of a message is stored in the offset data block. The addresses of the parameter data blocks are stored in continuous data blocks and are attached directly to the corresponding offset data blocks.
Relationship between the offset and parameter data blocks:

![Diagram showing the relationship between offset and parameter data blocks.]

Figure 3
<table>
<thead>
<tr>
<th>Category</th>
<th>Max. Number</th>
<th>Size of the Parameter Block</th>
<th>No. of Blocks per Parameter DB</th>
<th>Max. No. of Parameter DBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1008 / 2016</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1008</td>
<td>2 DWs</td>
<td>128</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1008</td>
<td>5 DWs</td>
<td>51</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>1008</td>
<td>7 DWs</td>
<td>36</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 8

Calculating the number of parameter data blocks:

\[
\text{Number of Parameters Data Block} = \frac{\text{used Messages}}{\text{Number of parameter blocks per parameter data block}}
\]

When configuring, care must be taken to ensure that addresses do not overlap with the data blocks of another message category and that the number of parameter data blocks can cope with any future upgrades.

A parameter data block contains parameter blocks which are assigned to the different messages. The parameter blocks are stored continuously in the parameter data block, starting with the parameter block for the first message of that message category. The parameter blocks are incremented continuously beyond the limits of the parameter DB. Upon reaching the end of the parameter DB, the parameter block is continued with the next number of the following parameter DB from DW 0 onwards. Only whole parameter blocks are stored in the parameter data block.

Calculating the start address of a parameter block:

\[
\text{Offset Message Number} = \text{Message Number - Basic Message Number}
\]

\[
\text{Parameter DB} = \text{Offset DB} + 1 + (\text{Offset Message Number} / \text{Parameter blocks per Parameter DB})
\]

\[
\text{Start Address of the Parameter DB} = (\text{Offset Message Number} \% \text{Parameter Blocks per Parameter DB}) \times \text{Size of the Parameter Block}
\]

The user must ensure that the corresponding data (process tags, job number, batch identifier) is available at the corresponding address.
5.2.3.11 Structure of a Message Block

A message block that is sent to the higher-level WinCC system consists of several consecutive data words. The data words contain message-specific information. The sum of the data words result in one message block. The size of the message blocks differs among the individual message categories.

Regardless of the message category, a message block always consists of at least two data words. These are the message number and the message status data words. Depending on whether the messages are date/time stamped (3 data words) and have been furnished with the appropriate parameters, a message block may be as long as 12 data words.

<table>
<thead>
<tr>
<th>DW</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st DW</td>
<td>Message Number</td>
</tr>
<tr>
<td>2nd DW</td>
<td>Message Status</td>
</tr>
<tr>
<td>3rd DW</td>
<td>Time</td>
</tr>
<tr>
<td>4th DW</td>
<td>Time</td>
</tr>
<tr>
<td>5th DW</td>
<td>Date</td>
</tr>
<tr>
<td>6th DW</td>
<td>Process Tag</td>
</tr>
<tr>
<td>7th DW</td>
<td>Process Tag</td>
</tr>
<tr>
<td>8th DW</td>
<td>Job Number</td>
</tr>
<tr>
<td>9th DW</td>
<td>Job Number</td>
</tr>
<tr>
<td>10th DW</td>
<td>Batch Identifier</td>
</tr>
<tr>
<td>11th DW</td>
<td>Reserve</td>
</tr>
<tr>
<td>12th DW</td>
<td>Reserve</td>
</tr>
</tbody>
</table>

Table 9

If messages are not date/time stamped, the three data words required for them at the provided third to fifth position of the block are omitted. The parameter data words are then attached without a gap to the status data word. The particular size of a message block (number of DWs) differs depending on the message category and desired date/time stamp and can be taken from table 10.

Determining the message block length as a function of the message category:

<table>
<thead>
<tr>
<th>Category</th>
<th>Message Block Length in DWs without Date and Time</th>
<th>Message Block Length in DWs with Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 10

5.2.3.12 Message Number

Every message is assigned a certain message number by which it can be clearly identified.
5.2.3.13 Message Status

The message status is structured as follows:

<table>
<thead>
<tr>
<th>DW</th>
<th>Message Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW</td>
<td>Time</td>
</tr>
<tr>
<td>DW</td>
<td>Date</td>
</tr>
<tr>
<td>DW</td>
<td>Process Tag</td>
</tr>
<tr>
<td>DW</td>
<td>Job Number</td>
</tr>
<tr>
<td>DW</td>
<td>Batch Identifier</td>
</tr>
<tr>
<td>DW</td>
<td>Reserve</td>
</tr>
</tbody>
</table>

Table 11

5.2.3.14 Date/Time Stamp

The date and the time are made available by the function block FB 86: MESS:CLOCK in binary code.

5.2.3.15 Process Tag

Two data words by which process tags can be recorded and forwarded to the process system, if a message comes in.

5.2.3.16 Job Number/Batch Identifier

Depending on the configuration, the first two data words have to be interpreted either as a signed 32-Bit binary number or as a total of four ASCII characters. The third data word has to be interpreted as two ASCII characters.

Through these three data words, the current job number or batch identifier can be transferred to the WinCC system, if a message comes in.

5.2.3.17 Reserve

The two reserve data words of the message category 4 are intended for future expansions, but are currently not yet implemented in the WinCC system.

5.2.3.18 Generation of a Message Block

After a message has been detected, the currently checked Bit position is used to determine the corresponding message number, which is stored as the first data word of the message block in the FIFO buffer. Depending on whether the message is coming in or going out, the category and the requirement for a date/time stamp, the corresponding status mask is selected and stored as the second data word of the message block in the FIFO buffer. If the parameters have been assigned to the corresponding Bit in the system data block for a date/time stamp, the three data words that are available in system data block 80 - starting from address DW 190 - will follow in the requested PC format. Depending on the message category, the associated parameter block is read, if required, from the corresponding data archive (parameter data block) and added to the last input in the FIFO buffer to complete the message block.
Then, the next status Bit of the message that follows is examined. This is continued until all messages to which parameters have been assigned have been processed.

5.2.3.19 The Internal FIFO Buffer (Ring)

A FIFO buffer is a type of storage that is shaped like a ring, i.e. the end of the storage ring is followed by its beginning. This results in the storage being limited in size on the one hand, but not being finite on the other due to its continuous restarting once the storage is filled.

In the message acquisition system this means that the oldest data is going to be overwritten by the most recent data once the virtual storage end is reached (buffer is full) - if the previous data is not removed - therefore resulting in the information being lost. The FIFO buffer in RAM acts, as its name suggests, as a buffer for the acquired messages before they are forwarded to the PC. In RAM, the FIFO buffer consists of a memory area of at least two data blocks and can, depending on the parameter assignment, be configured to any size as long as it is within the maximum number of permissible data blocks in a PLC or the remaining number available DBs of the application program. The user communicates to the alarm system the number of data blocks available to it for archiving. With more than one data block, it is imperative to use data blocks with continuous DB numbers. Consequently, the user specifies the start DB number and the end DB number of the buffer as parameters in the system DB. All data blocks whose values lie between the start data block and the end data block (including the two data blocks themselves) belong to the buffer as storage space.

5.2.3.20 The Send Mailbox - Data Transfer to the Higher-Level WinCC System

Generally, all message entries of a current cycle are written to the internal FIFO buffer of the S5 alarm system at first. The message entries (up to a maximum of the content of one data block) are transferred to the message interface (send mailbox) upon the completion of the acquisition, providing the send mailbox is ready. The message interface, in the form of a data block, acts as a data source for the transfer function blocks (STEP 5 - data handling blocks). The data handling blocks form an interface to the corresponding communication processor for the process bus being used (e.g. for the SINEC-H1 bus).

Structure of the send mailbox:

<table>
<thead>
<tr>
<th>DW</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW 0</td>
<td>Length of the Data Block</td>
</tr>
<tr>
<td>DW 1</td>
<td>KY = [ PLC No. ], [ CPU No. ]</td>
</tr>
<tr>
<td>DW 2</td>
<td>KY = [ 0 ], [ Number of Messages ]</td>
</tr>
<tr>
<td>DW 3</td>
<td>Start of the User Data (Message Blocks)</td>
</tr>
</tbody>
</table>

Table 12
DW 0:

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13

DW 0 of the send mailbox is determined first by the Bit number 14 - the activation edge trigger of a requested job - and secondly by the Bit numbers 0 - 8 - the source data length. Since the data block to be transferred can have a maximum length of 256 data words, but one Byte can only represent a number up to 255, a separate query of the Bytes using the DL or DR commands is not possible. It is therefore recommended to transfer the DW 0 be in an auxiliary flag. This also has the advantage that the Enable Bit can be evaluated separately and directly. This operation cannot be used for the application of data words.

If the condition is met, the Bit used as an edge for the one-time triggering of a send job should be reset. The remaining set Bits then correspond to the transferred source data length and can be written to the data area of the indirect parameter assignment as QLAE.

Following a successfully completed WRITE request (SINEC-H1) to the WinCC system (free of faults (FOF)), the DW 0 of the send mailbox must be overwritten with the value 0. This re-enables the send mailbox and additional message blocks, if present, can be transferred from the internal FIFO buffer to the send mailbox.

The WRITE request (SINEC-H1) has to be implemented using the SEND direct function. Information about this function can be found in the corresponding PLC manual.
5.2.4 Interface Description

The following interfaces and blocks will be described:

- **System Data Block DB 80**: For assigning parameters to the S5 alarm system.
- **Offset Data Block for the corresponding Message Category**: Binary interface of the message signals to the S5 alarm system with specification of the message properties.
- **Parameter Data Block for the corresponding Message Category**: For specifying additional message data of the categories 2 to 4.
- **Send Mailbox**: Transfer interface to the WinCC system.

5.2.4.1 System Data Block 80

By using the system data block DB 80, it is possible to configure independent data areas for four message categories, a FIFO storage and a send mailbox. For the configuration, the data words 0 to 20 in the DB 80 are provided.

5.2.4.2 Offset Data Block

The S5 alarm system evaluates the signal states of the corresponding messages and forms the corresponding message blocks from them, if required.

The user has to ensure that...

- the individual messages are specified during configuration of the idle states.
- the message states are written to the corresponding Signal Status Bits during the runtime of the S5 application program.
- the corresponding Acknowledgment Bits are read and evaluated, if required.

5.2.4.3 Parameter Data Block

For the message categories 2 to 4, additional information about the current status of the system can be transferred by the message block.

The user has to ensure that...

- upon the arrival of a message, the valid process tags (process value, job and batch numbers) are located in the corresponding parameter blocks.

5.2.4.4 Send Mailbox/Transfer Mailbox

As soon as it contains message blocks, the send mailbox is transferred directly to the WinCC system through a WRITE job (SINEC-H1).

The user has to ensure that...

- the corresponding data handling blocks of the respective CPU are available.
- appropriate communication channels for a process bus connection are specified during the configuration of the WinCC system.
- a WRITE job is triggered.
5.2.5 Assigning Parameters to the S5 Alarm System/System DB 80

Description of the configurable data words of the system data block DB 80:

<table>
<thead>
<tr>
<th>DW</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DB Address: internal FIFO start</td>
</tr>
<tr>
<td>1</td>
<td>DB Address: internal FIFO end</td>
</tr>
<tr>
<td>2</td>
<td>0: without date and time, 1: with date and time</td>
</tr>
<tr>
<td>3</td>
<td>DB offset for category 1 messages</td>
</tr>
<tr>
<td>4</td>
<td>1: one DB offset of category 1, 2: two DB offsets of category 1</td>
</tr>
<tr>
<td>5</td>
<td>DB offset for category 2 messages</td>
</tr>
<tr>
<td>6</td>
<td>DB offset for category 3 messages</td>
</tr>
<tr>
<td>7</td>
<td>DB offset for category 4 messages</td>
</tr>
<tr>
<td>8</td>
<td>Reserve</td>
</tr>
<tr>
<td>9</td>
<td>Reserve</td>
</tr>
<tr>
<td>10</td>
<td>DB Address: send mailbox CPU -&gt; PC</td>
</tr>
<tr>
<td>11</td>
<td>1: acquisition optimized (ACOP)</td>
</tr>
<tr>
<td>12</td>
<td>ACOP starting from n messages</td>
</tr>
<tr>
<td>13</td>
<td>PLC type (115/135/155)</td>
</tr>
<tr>
<td>14</td>
<td>Reserve (must be 1)</td>
</tr>
<tr>
<td>15</td>
<td>PLC No.: 1 to 255; CPU No.: 1 to 4</td>
</tr>
<tr>
<td>16</td>
<td>Reserve</td>
</tr>
<tr>
<td>17</td>
<td>Reserve</td>
</tr>
<tr>
<td>18</td>
<td>Reserve</td>
</tr>
<tr>
<td>19</td>
<td>Reserve</td>
</tr>
<tr>
<td>20</td>
<td>Parity error of the plausibility check</td>
</tr>
</tbody>
</table>

Table 14

DW 0, DW 1: DB memory area of the internal FIFO buffer
The internal FIFO buffer area for messages is defined by these two data words.
The storage space must have a size of at least two data blocks and it must be ensured that
the FIFO end is parameterized larger than the FIFO start.
The memory area of the buffer memory results from the data block area - including the two
specified data blocks - limited by the FIFO start and FIFO end.
Selection of the FIFO Buffer Size:
When the storage capacity of the FIFO buffer is reached, the oldest messages are
overwritten. The number of DBs must be selected large enough so that in the event of a
messages surge, no messages are overwritten before they can be exported. To ensure this,
the following rule of thumb is applied:
Determination of the Number of DBs in the FIFO Buffer:
Messages per DB = (255 DW / DB) / Message Block Length

See Table 10

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
|    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |

<table>
<thead>
<tr>
<th>Triggering Edge</th>
<th>Length of the Data Block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For ACOP operation of the message acquisition system, it is recommended to add one or two more data blocks.

**DW 2: Date and Time Identifier**
The option of date/time stamping messages refers to all parameterized messages. Either all the messages which have to be acquired are date/time stamped (DW2 = 1) or none (DW2 = 0). If DW2 = 0 is set, the WinCC system adds a date/time stamp to message blocks coming in.

**DW 3, DW 4: Offset DB of the Message Category 1**
If category 1 messages (messages without parameters and batch identifier) have to be configured, the address of the offset data block has to be specified in data word 3. The signal states of these messages have to be written continuously by the control program into the data blocks specified.

If more than 1008 messages (not more than 2016 messages) of category 1 are planned, an additional data block is enabled for category 1 messages by entering 2 in the data word 4. The second data block automatically receives the next higher address in relation to the address in DW 3. If a maximum of 1008 category 1 messages suffices, a 1 is entered in DW 4.

**DW 5, DW 6, DW 7: Offset DB of the Message Categories 2, 3, 4**
Similar to DW 3, data words 5 - 7 contain the data block addresses where the signals of the messages are stored.

DW 5 contains the address of the data block for message category 2, whereas DWs 6 and 7 contain the addresses for the message categories 3 and 4, respectively.

If a message category is not used, a 0 has to be entered in the corresponding DW.

The addresses specified in DWs 5 - 7 are so-called offset DBs. Depending on the message category and the number of messages per category, they are assigned a corresponding number of secondary DBs. The secondary DBs contain the parameters of the messages. For this reason, it must be ensured that - when assigning the offset DB addresses - there is sufficient space (data blocks) allocated for the parameter DBs between the previous offset DB and the one to be specified.

Up to 1008 messages can be configured for message categories 2 to 4. When fully utilized, a different number of secondary DBs (parameter DBs) to offset DBs arises for the different categories (see Table 8).

**DW 10: Message Interface to the Higher-Level WinCC System**
Parameters must always be assigned to the data word 10 - the operating mode used by the message acquisition system has no bearing on this. The DB address of the transfer mailbox is assigned in DW 10. The transfer mailbox acts as an interface between the SIMATIC S5 and the higher-level WinCC system.

**DW 11, DW 12: Mode Selection for optimum Acquisition and corresponding Number of Messages**
Two operating modes are available:
- 0 in DW 11 -> Normal mode of the message acquisition system
- 1 in DW 11 -> optimum acquisition mode of the message acquisition system

**Normal Mode:**
All messages that have been acquired and stored in the internal buffer within a cycle are sent to the WinCC station by the message interface (which has a limited capacity), provided that the message interface is ready to accept data.

This course of action results in a relatively long cycle time if a large number of messages come in within a cycle or within several consecutive cycles. The cycle time also increases with the message block size of the message categories involved. In this case, the acquisition of the message blocks involves more effort and takes longer.
Optimized Acquisition:

The chronological acquisition of the messages issued has priority over the sending to the WinCC station. Attention is focused on the relative time between the issuance of the system messages. The fact that messages might be displayed on the WinCC station a few milliseconds later is of secondary importance. The sluggishness of the human eye and the receptiveness of the observer in the control room are decisive factors.

In order to reduce the cycle time of the message acquisition system in such time-critical cases, the option of running the system in the optimized acquisition mode has been introduced. The minimum number of messages issued within an OB1 cycle is specified in DW 12. If the number of messages exceeds this minimum number during the current OB1 cycle, the messages are only acquired and buffered. They are not paged out or subsequently sent to a communication partner in this OB1 cycle.

DW 15: PLC/CPU Number

This data word is required for generating the message header and needs the specification of the project-related PLC number and its CPU number. The CPU number is particularly important if several CPUs are operating within a single PLC. Only in conjunction with the data word containing the ID for the messages can the higher-level WinCC system interpret sent data as a message, assign message-specific message texts and evaluate them accordingly.

The DW 15 is the only data word to receive the S5 data format KY during the configuration, i.e. two Bytes can be represented separately (separated by a comma). The left Byte contains the PLC number, which may be between 1 and 255. In the right Byte, the CPU number is specified, which may be between 1 and 4.

Example:

\[
\begin{align*}
KY & = 10,2 \\
PLC\ Number & = 10 \\
CPU\ Number & = 2
\end{align*}
\]

DW 20: Parameter Assignment Errors

All data words parameterized in the system DB are checked for their plausibility at the startup of the S5 alarm system. In this case, distinctions are made between exceeding possible value ranges, overlappings or multiple assignment of parameterized data blocks and missing specifications.

As the output parameter in the format of a data word, this function block uses a so-called PAFE word (Parameter Error word); it is similar to the system-specific data handling blocks. The status of the PAFE word can be taken from DW 20 in the system DB 80. The PAFE word can be examined for errors which may have occurred after the program return from FB 81. Following that, appropriate actions can be taken.

It is recommended to have the PLC jump to the stop status in the event of a PAFE word other than zero. If the PAFE word is ignored, no guarantee can be given for an error-free execution of the program.

Evaluation of the PAFE Word

If the program or PLC is brought to its stop status - as recommended - following the occurrence of an error (PAFE word unequal to 0), the error can be analyzed and corrected specifically by means of the error number. The following table provides information about the type of error caused during the parameterization.
Format of the PAFE Word:

\[ KY = \text{Error Number, Group Error ID} \]

Example:

\[ KY = 9,1 \]

The parameterization error with the number 9 corresponds to:
Offset DB address of category 1 is larger than the maximum permissible DB address.

<table>
<thead>
<tr>
<th>Error No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start DB of the int. buffer is not defined</td>
</tr>
<tr>
<td>2</td>
<td>Start DB of the int. buffer has the same address as the system DB (80)</td>
</tr>
<tr>
<td>3</td>
<td>Start DB addr. of the int. buffer is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>4</td>
<td>End DB of the int. buffer has the same address as the system DB (80)</td>
</tr>
<tr>
<td>5</td>
<td>End DB addr. is smaller than the start DB addr. of the int. buffer</td>
</tr>
<tr>
<td>6</td>
<td>End DB addr. of the int. buffer is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>7</td>
<td>Offset DB of category 1 has the same address as the system DB (80)</td>
</tr>
<tr>
<td>8</td>
<td>Offset DB address of category 1 is within the int. buffer area</td>
</tr>
<tr>
<td>9</td>
<td>Offset DB address of category 1 is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>10</td>
<td>Offset DB of category 2 has the same address as the system DB (80)</td>
</tr>
<tr>
<td>11</td>
<td>Offset DB of category 2 has the same address as that of category 1</td>
</tr>
<tr>
<td>12</td>
<td>Offset DB of category 2 has the same address as the 2nd offset DB of category 1</td>
</tr>
<tr>
<td>13</td>
<td>Offset DB address of category 2 is within the int. buffer area</td>
</tr>
<tr>
<td>14</td>
<td>Offset DB address of category 2 is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>15</td>
<td>Offset DB of category 3 has the same address as the system DB (80)</td>
</tr>
<tr>
<td>16</td>
<td>Offset DB of category 3 has the same address as that of category 1</td>
</tr>
<tr>
<td>17</td>
<td>Offset DB of category 3 has the same address as the 2nd offset DB of category 1</td>
</tr>
<tr>
<td>18</td>
<td>Offset DB of category 3 has the same address as that of category 2</td>
</tr>
<tr>
<td>19</td>
<td>Offset DB address of category 3 is within the int. buffer area</td>
</tr>
<tr>
<td>20</td>
<td>Offset DB address of category 3 is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>21</td>
<td>Offset DB of category 4 has the same address as the system DB (80)</td>
</tr>
<tr>
<td>22</td>
<td>Offset DB of category 4 has the same address as that of category 1</td>
</tr>
<tr>
<td>23</td>
<td>Offset DB of category 4 has the same address as the 2nd offset DB of category 1</td>
</tr>
<tr>
<td>24</td>
<td>Offset DB address of category 4 is within the int. buffer area</td>
</tr>
<tr>
<td>25</td>
<td>Offset DB address of category 4 is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>Error No.</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>26</td>
<td>Offset DB of category 4 has the same address as that of category 2</td>
</tr>
<tr>
<td>27</td>
<td>Offset DB of category 4 has the same address as that of category 3</td>
</tr>
<tr>
<td>28</td>
<td>The PC send mailbox has the same address as the system DB (80)</td>
</tr>
<tr>
<td>29</td>
<td>The PC send mailbox is not defined (0)</td>
</tr>
<tr>
<td>30</td>
<td>The PC send mailbox address is within the int. buffer area</td>
</tr>
<tr>
<td>31</td>
<td>The PC send mailbox address is larger than the maximum permissible DB address</td>
</tr>
<tr>
<td>32</td>
<td>The PC send mailbox has the same address as the offset DB of category 1</td>
</tr>
<tr>
<td>33</td>
<td>The PC send mailbox has the same address as the offset DB of category 2</td>
</tr>
<tr>
<td>34</td>
<td>The PC send mailbox has the same address as the offset DB of category 3</td>
</tr>
<tr>
<td>35</td>
<td>The PC send mailbox has the same address as the offset DB of category 4</td>
</tr>
<tr>
<td>36</td>
<td>The PC send mailbox has the same address as the 2nd offset DB of category 1</td>
</tr>
<tr>
<td>37</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>38</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>39</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>40</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>41</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>42</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>43</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>44</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>45</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>46</td>
<td>Reserve DW 9 or Reserve DW 10 not equal to 0</td>
</tr>
<tr>
<td>47</td>
<td>Number of messages for the minimum limit of the selected operating mode with optimized acquisition is missing</td>
</tr>
<tr>
<td>48</td>
<td>PLC type is not defined</td>
</tr>
<tr>
<td>49</td>
<td>Reserve DW 14 not equal to 1</td>
</tr>
<tr>
<td>50</td>
<td>PLC No. for message header is not defined</td>
</tr>
<tr>
<td>51</td>
<td>CPU No. for message header is not defined</td>
</tr>
<tr>
<td>52</td>
<td>CPU No. is larger than allowed (1 to 4)</td>
</tr>
</tbody>
</table>

Table 15
5.2.6 Configuration Sample for the S5 Alarm System

Description

The S5 alarm system is to be configured for the following message categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition: Message Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1200 messages (from message number 10000 to 11199) Messages 11000 to 11199 are low active</td>
</tr>
<tr>
<td>2</td>
<td>No messages planned</td>
</tr>
<tr>
<td>3</td>
<td>11 messages (from message number 30000 to 30010)</td>
</tr>
<tr>
<td>4</td>
<td>No messages planned</td>
</tr>
</tbody>
</table>

All messages are to be date/time stamped.
A 135U with PLC No. 1 and CPU No. 1 is going to be used.

5.2.6.1 DB 80 Parameterization

<table>
<thead>
<tr>
<th>Category</th>
<th>Max. Number</th>
<th>Size of the Parameter Block</th>
<th>No. of Blocks per Parameter DB</th>
<th>Max. No. of Parameter DBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1008 / 2016</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1008</td>
<td>2 DWs</td>
<td>128</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1008</td>
<td>5 DWs</td>
<td>51</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>1008</td>
<td>7 DWs</td>
<td>36</td>
<td>28</td>
</tr>
</tbody>
</table>

DB 81 is used as the PC send mailbox.
With a uniform occurrence of the existing messages, the average message block length (with date and time) is:

\[
\frac{1200 \times 5 + 11 \times 10}{1200 + 11} = 5.05
\]

Assumption:
The S5 alarm system is to accommodate a message surge of 100 messages in one PLC cycle and to operate in the optimized acquisition mode beginning with 30 messages.

\[
5 \text{ DW/mess. } \times 100 \text{ mess. } = 500 \text{ DWs}
\]

\[
\frac{500 \text{ DW}}{256 \text{ DW/DB}} = 1.95 \text{ DBs}
\]

This results in four data blocks for the FIFO buffer, since one or two more data blocks have to be added for the optimized acquisition mode.
The FIFO buffer starts at the data block address 82, which results in an end address of DB 85 for the FIFO buffer.
To provide a reserve for any future expansions of the FIFO buffer, the offset data block of category 1 is in DB 88 and DB 89 (for more than 1008 category 1 messages).
The DB 90 becomes the offset data block of the message category 3. A parameter DB of the message category 3 can accommodate up to 51 parameter blocks; if the 11 blocks used are subtracted, this results in an expandability by 40 category 3 messages with only one parameter data block (DB 91).
<table>
<thead>
<tr>
<th>DW</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DB Address: internal FIFO start</td>
<td>82</td>
</tr>
<tr>
<td>1</td>
<td>DB Address: internal FIFO end</td>
<td>85</td>
</tr>
<tr>
<td>2</td>
<td>0: without date and time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: with date and time</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>DB offset for category 1 messages</td>
<td>88</td>
</tr>
<tr>
<td>4</td>
<td>1: one DB offset of category 1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2: two DB offsets of category 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DB offset for category 2 messages</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>DB offset for category 3 messages</td>
<td>90</td>
</tr>
<tr>
<td>7</td>
<td>DB offset for category 4 messages</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Reserve</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Reserve</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>DB Address: send mailbox CPU -&gt; PC</td>
<td>81</td>
</tr>
<tr>
<td>11</td>
<td>1: acquisition optimized (ACOP)</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>ACOP starting from n messages</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>PLC type (115/135/155)</td>
<td>135</td>
</tr>
<tr>
<td>14</td>
<td>Reserve</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>PLC No.: 1 to 255; CPU No.: 1 to 4</td>
<td>1, 1</td>
</tr>
<tr>
<td>16</td>
<td>Reserve</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>Reserve</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>Reserve</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>Reserve</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>Parity error of the plausibility check</td>
<td>0</td>
</tr>
</tbody>
</table>

Data block 100 is used by DW 10 to DW 20 for synchronizing the time.
Data block 101 is used by DW 0 to DW 255 for receiving commands.

### 5.2.6.2 Setup of the Data Blocks

Creation of the data blocks DB 81 - DB 85, DB 88 - DB 91 and DB 101 of DW 0 - DW 255.
Creation of the data block DB 100 of DW 0 - DW 20.

### 5.2.6.3 Initialization of the Offset Data Blocks

Message Category 1
DB 88 and DB 89 are provided for the message category 1. The DB 88 contains the messages numbered 10000 to 11007 and the DB 89 the messages numbered 11008 to 11199.
A total of 1200 category 1 messages are to be configured.
See the chapter Address of the last Signal Status Block
Offset Message Number = Message Number - Basic Message Number = 0 to 1199
1st Offset DB:

Address of the last signal status block: DW 252

2nd Offset DB:

Address of the last signal status block: DW 252

\[
\begin{align*}
1200 - 1008 &= 192 \\
192 / 16 &= 12 \\
192 \% 16 &= 0
\end{align*}
\]

Address of the last signal status block in the offset data block 2 = 12 * 4 = 48

DB 88:

<table>
<thead>
<tr>
<th>DW</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW 0</td>
<td>Not assigned</td>
<td></td>
</tr>
<tr>
<td>DW 1</td>
<td>Basic Message Number</td>
<td>10000</td>
</tr>
<tr>
<td>DW 2</td>
<td>Address of last DW</td>
<td>252</td>
</tr>
<tr>
<td>DW 3</td>
<td>Not assigned</td>
<td></td>
</tr>
</tbody>
</table>

DB 89:

<table>
<thead>
<tr>
<th>DW</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW 0</td>
<td>Not assigned</td>
<td></td>
</tr>
<tr>
<td>DW 1</td>
<td>Basic Message Number</td>
<td>11018</td>
</tr>
<tr>
<td>DW 2</td>
<td>Address of last DW</td>
<td>48</td>
</tr>
<tr>
<td>DW 3</td>
<td>Not assigned</td>
<td></td>
</tr>
</tbody>
</table>

See the chapter Offset Message Number/Signal States of the Messages

The messages 11000 to 11199 are low active.

Position of the idle status Bit of the message number 11000:

\[
\begin{align*}
Offset Message Number: &= 11000 - 10000 = 1000 \\
Start of the signal status block &= (Offset Message Number / 16 + 1) * 4 = (62 + 1) * 4 * DW 252 \\
Data word of the idle status Bit: &= DW 253 \\
Data Bit: &= Offset Message Number \% 16 = 8 \\
Data block: &= Offset Data Block = DB 88
\end{align*}
\]
Position of the idle status Bit of the message number 11000:

Offset Message Number: \( 11199 - 10000 = 1199 \)
Start of the signal status block: \( \frac{\text{Offset Message Number}}{16} + 1 \times 4 = (74 + 1) \times 4 = 300 - 252 = 48 \)
Data word of the idle status Bit: DW 49
Data Bit: Offset Message Number \( \% 16 = 15 \)
Data block: Offset Data Block + 1 = DB 89

The following idle status Bits have to be changed:

**DB 88:**

DW 253: set data Bits 8 through 15 to 1

**DB 89:**

DW 5, DW 9, DW 13 through DW 49: set data bits 0 through 15 to 1
Message Category 3
For the message category 3, DB 90 is provided as the offset data block with the messages 30000 to 30010 and DB 91 as the parameter data block.
A total of 11 category 3 messages are to be configured.
See the chapter Structure of the Parameter Data Block
Offset Message Number = Message Number - Basic Message Number = 0 to 10

**Offset DB:**

Address of the last signal status block 11 / 16 = 0
Address of the last signal status block 11 \( \% 16 = 11 \)
Address of the last signal status block \((0+1) \times 4 = 4\)
### DB 89:

<table>
<thead>
<tr>
<th>DW</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW 0</td>
<td>Not assigned</td>
<td></td>
</tr>
<tr>
<td>DW 1</td>
<td>Basic Message Number</td>
<td>30000</td>
</tr>
<tr>
<td>DW 2</td>
<td>Address of last DW</td>
<td>4</td>
</tr>
<tr>
<td>DW 3</td>
<td>Not assigned</td>
<td></td>
</tr>
</tbody>
</table>

All idle status Bits are 0.  
See the chapter Structure of the Parameter Data Block

**Parameter DB**

Message Number 30000:

\[
\text{Parameter DB} = 90 + 1 + 0 \div 51 = 91
\]

Message Number 30010:

\[
\text{Parameter DB} = 90 + 1 + 10 \div 51 = 91
\]

\[
\text{Start address of the corresponding parameter block} = (\text{Offset Message Number} \% \text{Parameter Blocks per Parameter DB}) \times \text{Size of the Parameter Block}
\]

Message number 30000:

\[
\text{Start address of the corresponding parameter block} = (0 \% 51) \times 5 = \text{DW}
\]

Message Number 30010:

\[
\text{Start address of the corresponding parameter block} = (10 \% 51) \times 5 = \text{DW 50}
\]

**DB 91:** Parameter Data Block 91 to Offset Data Block 90

<table>
<thead>
<tr>
<th>Message Number</th>
<th>Process Values</th>
<th>Job Number</th>
<th>Batch Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>30000</td>
<td>DW 0, 1</td>
<td>DW 2, 3</td>
<td>DW 4</td>
</tr>
<tr>
<td>30001</td>
<td>DW 5, 6</td>
<td>DW 7, 8</td>
<td>DW 9</td>
</tr>
<tr>
<td>30002</td>
<td>DW 10, 11</td>
<td>DW 12, 13</td>
<td>DW 14</td>
</tr>
<tr>
<td>30003</td>
<td>DW 15, 16</td>
<td>DW 17, 18</td>
<td>DW 19</td>
</tr>
<tr>
<td>30004</td>
<td>DW 20, 21</td>
<td>DW 22, 23</td>
<td>DW 24</td>
</tr>
<tr>
<td>30005</td>
<td>DW 25, 26</td>
<td>DW 27, 28</td>
<td>DW 29</td>
</tr>
<tr>
<td>30006</td>
<td>DW 30, 31</td>
<td>DW 32, 33</td>
<td>DW 34</td>
</tr>
<tr>
<td>30007</td>
<td>DW 35, 36</td>
<td>DW 37, 38</td>
<td>DW 39</td>
</tr>
<tr>
<td>30008</td>
<td>DW 40, 41</td>
<td>DW 42, 43</td>
<td>DW 44</td>
</tr>
<tr>
<td>30009</td>
<td>DW 45, 46</td>
<td>DW 47, 48</td>
<td>DW 49</td>
</tr>
<tr>
<td>30010</td>
<td>DW 50, 51</td>
<td>DW 52, 53</td>
<td>DW 54</td>
</tr>
</tbody>
</table>
5.2.7 Documentation of the SIMATIC S5 Command Blocks

Purpose and Function of the S5 Command Blocks

The software is used to process Bits, Bytes, words and double words in the SIMATIC S5 via a process bus (e.g. Industrial Ethernet). Via the process bus, it is only possible to address Byte or word values in the SIMATIC S5.

By default, the following operations can be executed:

• Data blocks (DB and DX), timers and counters have to be changed only as words.
• Flags, inputs, outputs, peripheries (P and Q) have to be changed only as Bytes.

The program package provides the required software functionality within the SIMATIC S5 to implement the following operations of the WinCC system via the given process bus:

• Set an initializing pulse for one OB1 cycle
• Set, reset, invert the Bit in DB/DX
• Set, reset, invert the Bit in flag
• Write left/right Bytes to DB/DX
• Write word/double word to DB/DX
• Write Byte/word to flag
• Write Byte/word to periphery
• Write Byte/word to extended periphery

The desired changes in the SIMATIC S5 are made available by the WinCC Control Center as a raw data tag via a data interface. The commands have to be sent to the S5 via this raw data tag. These commands are evaluated and executed directly in the S5 through the command interpreter FB 87 : EXECUTE.

This manual describes the application and handling of the S5 command blocks in the SIMATIC S5 environment. The user receives an overview of the function and data blocks used by the software, and the storage space required. This is followed by a detailed interface description of the existing data interface. A configuration sample is presented to provide additional help.

5.2.7.1 Listing of the Software Blocks

The S5 Command Blocks of the SIMATIC S5 software are located on the WinCC CD-ROM in the file named WINCC1ST.S5D.

This file contains the following function blocks for the S5 command blocks:

<table>
<thead>
<tr>
<th>FB</th>
<th>Name</th>
<th>Size in Bytes</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB 87</td>
<td>EXECUTE</td>
<td>152</td>
<td>Enables Bit, Byte, word and double word manipulation via the process bus</td>
</tr>
<tr>
<td>FB 88</td>
<td>OPCODE</td>
<td>399</td>
<td>Called by FB 87</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>551</td>
<td></td>
</tr>
</tbody>
</table>

Table 16

Additionally, a 512 Byte command data block is required.
5.2.7.2 Hardware Requirements

The function blocks specified in table 16 require the following hardware in order to be executed correctly:

<table>
<thead>
<tr>
<th>PLC</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLC 115U</td>
<td>CPU 943, CPU 944, CPU 945</td>
</tr>
<tr>
<td>PLC 135U</td>
<td>CPU 928A, CPU 928B</td>
</tr>
<tr>
<td>PLC 155U</td>
<td>CPU 946/947, CPU 948</td>
</tr>
</tbody>
</table>

5.2.7.3 Call Parameters of the FB 87: EXECUTE

The following describes the call parameters of the function block FB 87: EXECUTE.

<table>
<thead>
<tr>
<th>Name</th>
<th>Execute</th>
<th>Parameter</th>
</tr>
</thead>
</table>

BNR: Data block number of the command transfer interface
DBDX: Data source type for the command transfer interface
DB: Data source is a data block (DB)
DX: Data source is an extended data block (DX)
RIMP: Bit position for the initializing pulse
RIMP: Flag number, Bit number
5.2.8 Interface Description

The following interfaces and blocks will be described:
- Command Function Block FB 87
- Command Data Block: Command Transfer Interface to the SIMATIC S5

In the SIMATIC S5, the command interpreter (FB 87: EXECUTE) is called cyclically in the OB 1. The type and address of the command DB are transferred as parameters. When a command is pending, the Op code and four parameters are forwarded to the FB 88: OPCODE and executed directly. After a command has been executed, the command counter (DW 1) is decremented by one. The process of transferring the command and decrementing the command counter is repeated until all pending commands have been processed.

Details about the type and address of the data block have to agree in both the WinCC Control Center and the S5 program, and the data block has to be present in the S5. Available for selection is a DB or DX data block and its address (e.g. DX 234). The data block has to be opened by the user up to data word 255, since the data words 0 - 255 can be addressed in the data block specified.

The following syntax has been defined for commands stored in the command data block:

<table>
<thead>
<tr>
<th>DW</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>Number of commands to be executed</td>
</tr>
<tr>
<td>2</td>
<td>Op code of the first command</td>
</tr>
<tr>
<td>3</td>
<td>Parameter 1 (Op code 1)</td>
</tr>
<tr>
<td>4</td>
<td>Parameter 2 (Op code 1)</td>
</tr>
<tr>
<td>5</td>
<td>Parameter 3 (Op code 1)</td>
</tr>
<tr>
<td>6</td>
<td>Parameter 4 (Op code 1)</td>
</tr>
<tr>
<td>7</td>
<td>Op code of the second command</td>
</tr>
<tr>
<td>8</td>
<td>Parameter 1 (Op code 2)</td>
</tr>
<tr>
<td>9</td>
<td>Parameter 1 (Op code 2)</td>
</tr>
<tr>
<td>10</td>
<td>Parameter 2 (Op code 2)</td>
</tr>
<tr>
<td>11</td>
<td>Parameter 3 (Op code 2)</td>
</tr>
<tr>
<td>12</td>
<td>Parameter 4 (Op code 2)</td>
</tr>
<tr>
<td>13</td>
<td>Op code of the third command</td>
</tr>
<tr>
<td>14</td>
<td>Parameter 1 (Op code 3)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
The following describes the syntax of the implemented commands:

Transfer of Op code and parameters to the command DB

<table>
<thead>
<tr>
<th>Command</th>
<th>Op Code</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
<th>Parameter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Bit in DB</td>
<td>10</td>
<td>DB</td>
<td>DW</td>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>Reset Bit in DB</td>
<td>11</td>
<td>DB</td>
<td>DW</td>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>Invert Bit in DB</td>
<td>12</td>
<td>DB</td>
<td>DW</td>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>Set right Byte in DB</td>
<td>15</td>
<td>DB</td>
<td>DW</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Set left Byte in DB</td>
<td>16</td>
<td>DB</td>
<td>DW</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Write data word to DB</td>
<td>17</td>
<td>DB</td>
<td>DW</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Write double word to DB</td>
<td>18</td>
<td>DB</td>
<td>DW</td>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>Set Bit in DX</td>
<td>20</td>
<td>DX</td>
<td>DW</td>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>Reset Bit in DX</td>
<td>21</td>
<td>DX</td>
<td>DW</td>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>Invert Bit in DX</td>
<td>22</td>
<td>DX</td>
<td>DW</td>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>Set right Byte in DX</td>
<td>25</td>
<td>DX</td>
<td>DW</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Set left Byte in DX</td>
<td>26</td>
<td>DX</td>
<td>DW</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Write data word to DX</td>
<td>27</td>
<td>DX</td>
<td>DW</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Write double word to DX</td>
<td>28</td>
<td>DX</td>
<td>DW</td>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>Set flag Bit</td>
<td>30</td>
<td>MB</td>
<td>Bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset flag Bit</td>
<td>31</td>
<td>MB</td>
<td>Bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invert flag Bit</td>
<td>32</td>
<td>MB</td>
<td>Bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write flag Byte</td>
<td>35</td>
<td>MB</td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write memory word</td>
<td>36</td>
<td>MW</td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write periphery Byte</td>
<td>45</td>
<td>PB</td>
<td>Value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.2.8.1 Configuration Sample for the S5 Command Blocks

The S5 command blocks are to be set up. The initializing pulse is provided in the flag word 56, Bit 4. DX 237 serves as the command data block. Ensure that the data block DX 237 is opened from DW 0 to 255 in the PLC. In the WinCC Control Center, the desired data block is entered when specifying the channel parameters (e.g. Industrial Ethernet).

Excerpt from OB 1:

:SPA  FB 87
NAME :EXECUTE
DBNP :KF +237
DBDX :KC DX
RIMP :KY 56, 4
5.2.9 Task and Function of the S5 Time Synchronization

This document describes the functions and properties of the SIMATIC S5 software:

S5 Time Synchronization

The software is used to synchronize the SIMATIC S5 system clock. In addition, it supplies the proper date/time data format for the generation of the message blocks to the chronological message acquisition of the S5 alarm system.

The function block FB 86: MESS:CLOCK also provides the current S5 time in a format required by the chronological message acquisition. The data is located in the system data block 80 from DW 190 onward.

If a change in a message signal status occurs, the message is identified by the function block FB 80: SYSTEMFB via its message number and stamped with the current date/time from system data block 80.

This manual describes in detail the application and handling of the S5 time synchronization in the SIMATIC S5 environment. The user receives an overview of the function and data blocks used by the software, and the storage space required. A configuration sample is presented to provide additional help.

5.2.9.1 Listing of the Software Blocks

The SIMATIC S5 software (S5 time synchronization) is located on the WinCC CD-ROM in the file named WINCC1ST.S5D.

This file contains the following function and data blocks:

<table>
<thead>
<tr>
<th>FB</th>
<th>Name</th>
<th>Size in Bytes</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB 86</td>
<td>MESS:CLOCK</td>
<td>1135</td>
<td>Synchronization of time</td>
</tr>
</tbody>
</table>

Table 17

- Clock data area 115U: 27 DWs = 54 Bytes
- Clock data area 135U/155U: 12 DWs = 24 Bytes
- Data area for the S5 alarm system: 3 DWs = 6 Bytes

5.2.9.2 Hardware Requirements

The function blocks specified for the S5 alarm system require the following hardware in order to be executed correctly:

<table>
<thead>
<tr>
<th>PLC</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLC 115U</td>
<td>CPU 944 *, CPU 945</td>
</tr>
<tr>
<td>PLC 135U</td>
<td>CPU 928B</td>
</tr>
<tr>
<td>PLC 155U</td>
<td>CPU 946/947, CPU 948</td>
</tr>
</tbody>
</table>

* Only the CPU 944 with two PG interfaces has a system clock.
5.2.10 Call Parameters of the FB 86: MESS:CLOCK

The following describes the call parameters of the function block FB 86: MESS:CLOCK.

<table>
<thead>
<tr>
<th>Name</th>
<th>MESS:CLOCK</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID:</td>
<td>DCF7</td>
<td>E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF</td>
</tr>
</tbody>
</table>

CPUT:

<table>
<thead>
<tr>
<th>No. of the CPU</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU 943 / CPU 944</td>
</tr>
<tr>
<td>2</td>
<td>CPU 945</td>
</tr>
<tr>
<td>3</td>
<td>CPU 928B</td>
</tr>
<tr>
<td>4</td>
<td>CPU 946 / 947</td>
</tr>
<tr>
<td>5</td>
<td>CPU 948</td>
</tr>
</tbody>
</table>

DCF7:

Operating mode
0 = operation with S5 system clock
1 = operation with DCF77 radio clock

QTYPE:

Type of data source for the time synchronization message
0 = data source is a data block (DB)
1 = data source is an extended data block (DX)

QSYN:

Data source of the time data

DCF7 = 0: QSYN = DB number, DW number of the time synchronization message received
DCF7 = 1: QSYN = DB number, DW number of the DCF77 time
**UDAT:**
Address of the clock data area
UDAT = DB number, DW number

**ZINT:**
Time interval in minutes for the sending of the synchronization message (DCF7 = 1)

**ZCLOCK:**
Target data area for the time data in the alarm system format
ZCLOCK = DB number, DW number

**ZSYN:**
Target data area for the time synchronization message (DCF7 = 1)
If the chronological message acquisition functionality is to be used with the S5 alarm system, a special time data format is expected in DB 80 from DW 190 onward. This time data format is derived from the S5 system time and is written to the corresponding ZCLOCK data area (DB 80, DW 190 - 192).

Relationship between chronological reporting and FB 86: MESS:CLOCK:

![Diagram](image)

Figure 4
### 5.2.11 Data Formats for Date and Time

Time synchronization message from a system (WinCC currently does not support the time message).
The first data word of the time synchronization message contains a source ID, which is sent by the system together with the date and time data.
The function block FB 86: MESS:CLOCK extracts the pending message only after the source ID FFFF is positioned at this location. The receipt of the message is acknowledged by a 0 in this data word. Only after a new message arrives (DW 1 = FFFF), is it read and evaluated again.

<table>
<thead>
<tr>
<th>Description</th>
<th>Data Word</th>
<th>Content</th>
<th>Validity Range</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source ID/Time Message</td>
<td>1</td>
<td>FFFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message ID</td>
<td>2</td>
<td>FFFF</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>Seconds</td>
<td>3</td>
<td>00xx</td>
<td>xx: 0 to 59</td>
<td></td>
</tr>
<tr>
<td>Minutes</td>
<td>4</td>
<td>00xx</td>
<td>xx: 0 to 59</td>
<td></td>
</tr>
<tr>
<td>Hours</td>
<td>5</td>
<td>00xx</td>
<td>xx: 0 to 23</td>
<td></td>
</tr>
<tr>
<td>Day</td>
<td>6</td>
<td>00xx</td>
<td>xx: 1 to 31</td>
<td></td>
</tr>
<tr>
<td>Month</td>
<td>7</td>
<td>00xx</td>
<td>xx: 1 to 12</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>8</td>
<td>00xx</td>
<td>xx: 0 to 127 (1990-2117)</td>
<td>Year + 1990</td>
</tr>
<tr>
<td>Day of Week</td>
<td>9</td>
<td>00xx</td>
<td>xx: 0 to 6</td>
<td>Sunday = 0</td>
</tr>
<tr>
<td>Day of Year</td>
<td>10</td>
<td>00xx</td>
<td>xx: 1 to 365</td>
<td></td>
</tr>
<tr>
<td>Summer Time, Winter time</td>
<td>11</td>
<td>yyxx</td>
<td>xx: Winter Time = 00 Summer Time = 01</td>
<td></td>
</tr>
<tr>
<td>Leap Year</td>
<td></td>
<td></td>
<td>yy: Leap Year</td>
<td>Current Year = 00 Last Year = 01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Two Years Ago = 02 Three Years Ago = 03</td>
</tr>
</tbody>
</table>

Table 18
5.2.11.1 Clock Data Area CPU 944, CPU 945

The data word numbers are relative specifications. The actual position of the area is determined by the call parameters: UDAT = DB No., DW No. of FB 86: MESS:CLOCK.

<table>
<thead>
<tr>
<th>DW</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>internal tags</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>current time</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>time operating range</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>reserve - alarm time</td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>reserve - operating hours</td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>time / date</td>
</tr>
<tr>
<td>24</td>
<td>after RUN / STOP</td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
</tbody>
</table>

Table 19
Current time in the clock data area:

<table>
<thead>
<tr>
<th>DW</th>
<th>Word/Left</th>
<th>Word/Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>---</td>
<td>Day of Week</td>
</tr>
<tr>
<td>6</td>
<td>Day</td>
<td>Month</td>
</tr>
<tr>
<td>7</td>
<td>Year</td>
<td>AM/PM (Bit, No. 7), Hour</td>
</tr>
<tr>
<td>8</td>
<td>Minute</td>
<td>Second</td>
</tr>
</tbody>
</table>

Figure 5
Setting area in the clock data area:

<table>
<thead>
<tr>
<th>DW</th>
<th>Word/Left</th>
<th>Word/Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Leap Year</td>
<td>Day of Week</td>
</tr>
<tr>
<td>10</td>
<td>Day</td>
<td>Month</td>
</tr>
<tr>
<td>11</td>
<td>Year</td>
<td>AM/PM (Bit, No. 7)</td>
</tr>
<tr>
<td>12</td>
<td>Minute</td>
<td>Second</td>
</tr>
</tbody>
</table>

Figure 6

5.2.11.2 Clock Data Area CPU 928B, CPU 948

The data word numbers are relative specifications. The actual position of the area is determined by the call parameters: UD AT = DB No., DW No. of FB 86: MESS:CLOCK.

<table>
<thead>
<tr>
<th>DW</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>internal tags</td>
</tr>
<tr>
<td>1</td>
<td>current time</td>
</tr>
<tr>
<td>2</td>
<td>time operating range</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7

Current time in the clock data area:

<table>
<thead>
<tr>
<th>DW</th>
<th>Word/Left</th>
<th>Word/Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14 13 12 11 10 9 8 7</td>
<td>6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>4</td>
<td>Seconds</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>Format</td>
<td>Hours</td>
</tr>
<tr>
<td>6</td>
<td>Day of Month</td>
<td>Day of Week</td>
</tr>
<tr>
<td>7</td>
<td>Year</td>
<td>Second</td>
</tr>
</tbody>
</table>

Figure 8

Current time in the setting area:

<table>
<thead>
<tr>
<th>DW</th>
<th>Word/Left</th>
<th>Word/Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14 13 12 11 10 9 8 7</td>
<td>6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>8</td>
<td>Seconds</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Format</td>
<td>Hours</td>
</tr>
<tr>
<td>10</td>
<td>Day of Month</td>
<td>Day of Week</td>
</tr>
</tbody>
</table>

Figure 9
5.2.11.3 Clock Data Area CPU 946, CPU 947

The data word numbers are relative specifications. The actual position of the area is determined by the call parameters: UDAT = DB No., DW No. of FB 86: MESS:CLOCK.

<table>
<thead>
<tr>
<th>DW</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>internal tags</td>
</tr>
<tr>
<td>1</td>
<td>current time</td>
</tr>
<tr>
<td>2</td>
<td>time operating range</td>
</tr>
</tbody>
</table>

Figure 10

Current time in the clock data area:

<table>
<thead>
<tr>
<th>DW</th>
<th>Word/Left</th>
<th>Word/Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10 sec.</td>
<td>1/10 sec.</td>
</tr>
<tr>
<td>6</td>
<td>10 h</td>
<td>1 min.</td>
</tr>
<tr>
<td>7</td>
<td>10 Days</td>
<td>Day of Week</td>
</tr>
<tr>
<td>8</td>
<td>10 Years</td>
<td>1 Month</td>
</tr>
</tbody>
</table>

Figure 11

Current time in the setting area:

<table>
<thead>
<tr>
<th>DW</th>
<th>Word/Left</th>
<th>Word/Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>10 sec.</td>
<td>1/10 sec.</td>
</tr>
<tr>
<td>10</td>
<td>10 h</td>
<td>1 min.</td>
</tr>
<tr>
<td>11</td>
<td>10 Days</td>
<td>Day of Week</td>
</tr>
<tr>
<td>12</td>
<td>10 Years</td>
<td>1 Month</td>
</tr>
</tbody>
</table>

Figure 12
5.2.11.4 Clock Data Format for Message Blocks

The data word numbers are relative specifications. The actual position of the area is determined by the parameters ZCLOCK = DB No., DW No. of FB 86: MESS:CLOCK.

If the chronological message acquisition functionality is to be used with the S5 alarm system, the data DB 80, DW 190 has to be entered into the parameter ZCLOCK.

The date and time are made available in binary code by the function block FB 86: MESS:CLOCK for the message processing:

Current time in the setting area:

<table>
<thead>
<tr>
<th>Description</th>
<th>Data Word</th>
<th>Bit</th>
<th>Validity Range</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/100 sec.</td>
<td>1</td>
<td>0 - 6</td>
<td>0 to 99 (0 - 990 msec.)</td>
<td>In one 10 msec. Raster</td>
</tr>
<tr>
<td>Seconds</td>
<td>1</td>
<td>7 - 12</td>
<td>0 to 59</td>
<td></td>
</tr>
<tr>
<td>Minutes</td>
<td>0</td>
<td>0 - 5</td>
<td>0 to 59</td>
<td></td>
</tr>
<tr>
<td>Hours</td>
<td>0</td>
<td>6 - 10</td>
<td>0 to 23</td>
<td></td>
</tr>
<tr>
<td>Day</td>
<td>2</td>
<td>0 - 4</td>
<td>1 to 31</td>
<td></td>
</tr>
<tr>
<td>Month</td>
<td>2</td>
<td>5 - 8</td>
<td>1 to 12</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>2</td>
<td>9 - 15</td>
<td>0 to 127 (1990-2117)</td>
<td>Year + 1990</td>
</tr>
</tbody>
</table>

Figure 13

**DW3: Time**

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
</tr>
</tbody>
</table>

**DW4: Time**

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
</tr>
</tbody>
</table>

**DW4: Date**

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
</tr>
</tbody>
</table>

Figure 14
5.2.12 Interface Description

To use the S5 time synchronization software, the user has to:

- fill out the call parameters of the FB 86: MESS:CLOCK as described in the chapter Call Parameters of the FB 86: MESS:CLOCK
- open the data areas in the PLC

Configuration Sample

Let us assume a CPU 944 with two PG interfaces. On this CPU, the S5 time synchronization for the S5 alarm system without the DCF77 clock is to be set up.

Data Areas:

<table>
<thead>
<tr>
<th>Time Synchronization Message:</th>
<th>DB 100, DW 20 - DW 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Data Area of the S5 System Clock:</td>
<td>DB 100, DW 31 - DW 47</td>
</tr>
<tr>
<td>Message Block Data*:</td>
<td>DB 80, DW 190 - DW 192</td>
</tr>
<tr>
<td>*</td>
<td>The application of the SIMATIC S5 alarm system requires the use of this data area.</td>
</tr>
</tbody>
</table>

During the configuration of the channel parameters (e.g. Industrial Ethernet) in the system, the desired data block (DB 100, DW20 - DW 30) must be entered for the time synchronization entry.

Ensure that DB 80 of DW 0 to DW 255 and DB 100 of DW 0 to DW 47 are opened.

Excerpt from OB 1:

: SPA FB 86
NAME : MELD: UHR
CPU1 : KF +1
DCF7 : KF +0
QTYP : KF +0
QSYN : KY 100, 20
UDAT : KY 100, 31
ZINT : KF +0
ZUHR : KY 80, 190
ZSYN : KF +0
5.2.13 Interaction with the WinCC Alarm System

The following must be taken into account regarding the interaction of the WinCC alarm system with the S5 message block:
In the send block of the S5, 256 has to be specified as the number of data words to be transferred.
In the Control Center, a new driver connection must be set up for the S5 Transport channel unit. In the Connection tab, specify the read function fetch passive.
For the data exchange with the alarm system, two raw data tags must be created for each PLC.
The first is responsible for receiving messages.
Set its addressing as follows. Data Area: DB, DB No.: xx, Address: Word, DW: 0, Raw Data Type: Event
The second is required for sending acknowledgment information.
Set its addressing as follows. Data Area: DB, DB No.: 80, Address: Word, DW: 90, Raw Data Type: Event
In the alarm system, the event tag is connected to the receive raw data tag (in this case, the Bit information has no meaning).
The acknowledgment tag is connected to the send raw data tag (in this case, the Bit information has no meaning).
As the format DLL, the file S5STD.NLL is entered.
Tip: Through the interconnection Wizard, all concerned messages can be connected in one operation.
Only positive fixed-point numbers are valid as process values. Floating-point values are not supported.

<table>
<thead>
<tr>
<th>S5</th>
<th>WinCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>prozess value</td>
<td>prozess value 1</td>
</tr>
<tr>
<td>job description</td>
<td>prozess value 2</td>
</tr>
<tr>
<td>batch description</td>
<td>prozess value 3</td>
</tr>
<tr>
<td>reserve</td>
<td>prozess value 4</td>
</tr>
</tbody>
</table>
5.3 Format DLL Interface to Alarm Logging and Tag Logging

Goal
The Alarm Logging and Tag Logging applications acquire the process data through the WinCC data manager. Depending on the communication type to the process,
• different channel DLLs are involved with the data transfer
• the process data is stored in message (raw data tags) with different structures
However, Alarm Logging and Tag Logging are to process the process data in the same manner, independent of the communication type used. For this reason, a separate Format DLL is used for every communication type, which knows the precise structure of the messages concerned and, from there, derives a universal process data form for Alarm Logging and Tag Logging.
Basically, a format DLL belongs to a channel DLL - like a channel DLL, the format DLL should be simple to add or remove to and from the overall system. Still, it does not have a direct interface to the associated channel DLL.
This document describes the integration and the interface of every format DLL to the Alarm Logging and Tag Logging WinCC applications. The document originated while drafting the S7PMC Format DLL, because of this, the term S7PMC Format DLL is for the most part synonymous with the term Format DLL.

Basic Process
The S7PMC format DLL is a passive group of programs, which exclusively has interfaces to the Alarm Logging and Tag Logging applications. The S7PMC format DLL processes S7PMC-specific functions for Alarm Logging and Tag Logging.
Alarm Logging and Tag Logging log on to the format DLL using a start call. In this case, certain parameters are transferred in a start structure to the format DLL and their properties are received via IDs.
Two data transfer directions are required to process the S7PMC functions in runtime:
• OS to PLC: (sending of logon/logoff jobs, acknowledgments)
• PLC to OS: (receipt of messages and archive data)

Through an initialization call, Tag Logging/Alarm Logging communicates the configured archive tag names and message numbers to the S7PMC DLL. For these objects, the format DLL (WinCC) must log on at the PLC. The initialization call can be processed at any point in time.
The format DLL is called by Alarm Logging/Tag Logging for the de-initialization in order to return resources, etc.
5.3.1 Shared Interface to Alarm Logging and Tag Logging

The general functions of the format DLL, which are identical for Alarm Logging and Tag Logging, are grouped in a shared interface. The function names all start with NORM... (Prefix for Alarm Logging-specific functions: Mld..., prefix for Tag Logging-specific functions: Pde....)

MELD = Alarm Logging
PDE = Tag Logging

![Diagram of shared interface]

1) The language switch is only required for format DLLs that contain a dialog box.
Alarm Logging-specific Additions

Runtime

Expanded Configuration Dialog
5.3.2 Tag Logging-specific Additions

Runtime

Expanded Configuration Dialog
5.3.3 API Functions of a WinCC Format DLL

The format DLL is divided into the following subsections:

- Initialization of the Format DLL
- Initialization by the operating system while the format DLL is being loaded (LibMain)
- Query of the properties of a format DLL
- Query of the name of the format DLL
- Shut down of the Format DLL
- Shut down by Tag Logging and Alarm Logging
- Unloading by the operating system
- Expansions to the Configuration
- Dialog expansions during message configuration
- Dialog expansion during archive tag configuration
- Online Services
- Registration of all format DLL-specific objects (messages, archive tags)
- Language switch
- Formatting
- Formatting of messages
- Formatting of archive tags

5.3.3.1 Initialization of the Format DLL

Initialization during the Load Operation

The Alarm Logging and/or Tag Logging applications load a WinCC format DLL with the help of the LoadLibrary system call. Following that, the format DLL is loaded by the operating system and initialized via its standard mechanisms. All entry addresses of the format DLL are defined.
5.3.3.2 Query of the Properties of a Format DLL

Alarm Logging and Tag Logging log on to the relevant format DLL via the call NormDLLStart. It is intended for the exchange of information between the format DLL and the application.

NormDLLStart

```c
#include <winccnrn.h>

BOOL NormDLLStart(
    LPVOID lpUser,
    BOOL bModeRuntime,
    PNORM_STARTSTRUCT pcs,
    PCHN_ERROR lpError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpUser</td>
<td>Pointer pointing to the application data, forwarded unchanged to callback</td>
</tr>
<tr>
<td>bModeRuntime</td>
<td>TRUE if the format DLL is started in runtime mode, FALSE if it is started in configuration mode; is currently not evaluated by the format DLL</td>
</tr>
<tr>
<td>pcs</td>
<td>Pointer pointing to the start structure</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the default WinCC error structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>No error</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>

NORM_STARTSTRUCT

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwSize</td>
<td>Size of the structure in Bytes</td>
<td>O</td>
</tr>
<tr>
<td>lpstrProjectPath</td>
<td>Path of the currently selected project</td>
<td>I</td>
</tr>
<tr>
<td>NORM_SEND_PROC</td>
<td>Pointer pointing to the callback function of the application, through which the format DLL sends a raw data tag - via the Data Manager - to the PLC.</td>
<td>I</td>
</tr>
<tr>
<td>pfWriteRwData</td>
<td>Application ID: 1 = Alarm Logging 2 = Tag Logging 3 = USER (reserved for future applications, currently not used)</td>
<td>I</td>
</tr>
<tr>
<td>dwAppID</td>
<td>Language setting at the time of call</td>
<td>I</td>
</tr>
<tr>
<td>dwNormCap</td>
<td>Properties of the format DLL according to the table below</td>
<td>O</td>
</tr>
</tbody>
</table>
The callback function for sending raw data tags to the WinCC Data Manager is supplied as follows:

```c
typedef BOOL(WINAPI* NORM_SEND_PROC)(
    LPDM_VAR_UPDATE_STRUCT lpDmVarUpdate,
    DWORD dwWait,
    LPVOID lpUser,
    LPCOMM_ERROR lpError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpDmVarUpdate</td>
<td>Pointer pointing to the raw data tag</td>
</tr>
</tbody>
</table>
| dwWait                  | Identification of whether the application should wait until completion of the write call or not:  
                           | WAIT_ID_NO with SET_VALUE  
                           | WAIT_ID_YES with SET_VALUE_WAIT                                        |
| lpUser                  | Pointer pointing to the application data, noted upon the call NormDLLStart |
| lpError                 | Pointer pointing to the default WinCC error structure                       |

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>No error</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>
A Bit is assigned to each property according to this table:

<table>
<thead>
<tr>
<th>DEFINE</th>
<th>Bit Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMCAP_DIALOG</td>
<td>0x00000001</td>
<td>Set Format DLL provides a special dialog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Format DLL does not provide a dialog</td>
</tr>
<tr>
<td>NORMCAP_REENTRANT</td>
<td>0x00000002</td>
<td>Set Format DLL can be reentered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Format DLL cannot be reentered</td>
</tr>
<tr>
<td>NORMCAP_MSG_FREE_LOCK</td>
<td>0x00000004</td>
<td>Set Logon/logoff is possible for messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Logon/logoff is not possible for messages</td>
</tr>
<tr>
<td>NORMCAP_ARC_FREE_LOCK</td>
<td>0x00000008</td>
<td>Set Logon/logoff is possible for archive tags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Logon/logoff is not possible for archive tags</td>
</tr>
<tr>
<td>NORMCAP_MSG_GENERIC</td>
<td>0x00000010</td>
<td>Set Messages can be generated generically</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Messages cannot be generated generically</td>
</tr>
<tr>
<td>NORMCAP_ARC_GENERIC</td>
<td>0x00000020</td>
<td>Set Archive tags can be generated generically</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Archive tags cannot be generated generically</td>
</tr>
</tbody>
</table>

### 5.3.3.3 Query of the Name of the Format DLL

**NormGetDLLName**

```c
#include <wincccur.h>
LPTSTR NormGetDLLName( void );
```

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTSTR</td>
<td>Pointer pointing to a string, which contains the name of the format DLL in plain text; the name depends on the current language setting.</td>
</tr>
</tbody>
</table>
5.3.4 Shut Down of the Format DLL

Shut Down by Tag Logging and Alarm Logging

Tag Logging and Alarm Logging notify the format DLL, if the applications are being closed. The resources are then returned properly in the format DLL.

NormDLLStop

\[\text{\#include } \langle \text{winccnhr.h} \rangle\]

\[\text{BOOL NormDLLStop (void)};\]

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Function successful</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function</td>
</tr>
</tbody>
</table>

Unloading by the Operating System

No special precautions have to be taken.

5.3.4.1 Expansions to the Configuration

Certain specifications are required for the S7PMC objects. These specifications are first requested in a dialog using standard means (without MFC) and go directly into either the WinCC message number or the name of the archive tag. This means that the format DLL does not have to store and manage these specifications itself. To guarantee the project-wide uniqueness of a message number or archive tag, an assignment between the message number or archive tag and the associated raw data tag is necessary. This assignment information is an integral part of the message number or archive tag name.

5.3.4.2 Dialog Expansion during the Configuration of S7PMC Messages

The format DLL has an API function for defining the S7PMC-specific message number. This function is called by the CS alarm system while assigning parameters to single messages belonging to a S7PMC format DLL. The message number assigned by the S7PMC format DLL is a number consisting of two parts.
Part 1:

The number which uniquely identifies a PLC CPU project-wide (raw data tag number).

Part 2:

The number belonging to the message from the PLC side that uniquely identifies the message within a PLC CPU (format DLL-specific).

In the configuration dialog, the following selection has to be made to establish the message number:

Structure of an S7PMC message number (32 Bit)

<table>
<thead>
<tr>
<th>Segment 1</th>
<th>Segment 2</th>
</tr>
</thead>
</table>

To Part 1

Every message belongs to a raw data tag, which identifies a PLC CPU. In order to perform the assignment of the raw data tag to the message number, the following definition has been established.

The name of the raw data tag for the S7PMC - and all connection types with a format DLL - has the following fixed structure:

@rd_alarm#rd_nr

@rd_alarm# Fixed integral part of the name of a raw data tag for format DLLs
rd_nr Decimal number between 0 and 1023 for the identification of a raw data tag (without leading zeros)

The most significant Bit of the message number is set for message numbers which are assigned by format DLLs (externally). These messages must only be processed by the corresponding format DLLs, i.e. via the configuration dialog of Alarm Logging, the message number cannot be changed.
To Part 2

This part of the message number can be assigned by the respective format DLL. For the S7PMC, it has the following meaning:

<table>
<thead>
<tr>
<th>MKI</th>
<th>Message class; one of the following classes has to be selected:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCAN (1)</td>
</tr>
<tr>
<td></td>
<td>ALARM/NOTIFY (2)</td>
</tr>
<tr>
<td></td>
<td>ALARM_8P/ALARM_8 (2)</td>
</tr>
<tr>
<td></td>
<td>LTM (3)</td>
</tr>
<tr>
<td>Sub-No.</td>
<td>Submessage number applicable only to ALARM_8 and ALARM_8P:</td>
</tr>
<tr>
<td></td>
<td>1...8</td>
</tr>
<tr>
<td>PMC-ID</td>
<td>PMC message number (block input parameter EV-ID):</td>
</tr>
<tr>
<td></td>
<td>1...16386</td>
</tr>
<tr>
<td></td>
<td>for the message classes SCAN and ALARM/NOTIFY or</td>
</tr>
<tr>
<td></td>
<td>ALARM_8P/ALARM_8</td>
</tr>
<tr>
<td></td>
<td>1...7</td>
</tr>
<tr>
<td></td>
<td>for the message class LTM</td>
</tr>
</tbody>
</table>

MldShowDialog

#include <wincntrn.h>

BOOL WINAPI MldShowDialog(
    HWND hwnd,                  // Window handle
    LPCSCREATEMSGDATA_GENERIC lpmCS,  // Pointer pointing to the single message data
    LPM_PROJECT_INFO lpDMProjectInfo,  // Pointer pointing to the project information structure
    LPCMH_ERROR lpError          // Pointer pointing to the default WinCC error structure
);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwnd</td>
<td>Window handle</td>
</tr>
<tr>
<td>lpmCS</td>
<td>Pointer pointing to the single message data</td>
</tr>
<tr>
<td>lpDMProjectInfo</td>
<td>Pointer pointing to the project information structure</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the default WinCC error structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Function successful</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>
5.3.4.3 Dialog Expansion during the Configuration of Archive Tags

The Format DLL has an API function for defining the S7PMC-specific archive tag name. This function is called by CS Tag Logging while assigning parameters to the archive tags belonging to an S7PMC connection. The archive tag name assigned by the S7PMC format DLL is a name consisting of several components that contains, among other things, the number belonging to the archive on the PLC. With this algorithm, the S7PMC tag numbers are uniquely contained in the WinCC archive tag description, resulting in the quickest possible assignment during runtime.

Tag Logging guarantees that the archive tag names are all unique.

Structure of an S7PMC archive tag name (not longer than 18 Bytes)

<table>
<thead>
<tr>
<th>Name</th>
<th>Length in Bytes</th>
<th>Assigned By</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>9</td>
<td>Tag Logging</td>
<td>Fixed character string assigned by Tag Logging consisting of the format DLL name and # as the separator, e.g. for the S7PMC: NRMS7PMC, not displayed by the interface</td>
</tr>
<tr>
<td>rw_id</td>
<td>8</td>
<td>Tag Logging/Format DLL</td>
<td>Raw data ID in hexadecimal character format (including leading zeros) resulting in a unique assignment to the raw data tag (connection) to which the archive number belongs. The name portion is formed by the format DLL using the Tag Logging input parameters.</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>Format DLL CS Portion</td>
<td>S7PMC-specific ID for differentiating between BSEND and AR_SEND: A = AR_SEND B = BSEND</td>
</tr>
<tr>
<td>ar_id</td>
<td>4</td>
<td>Format DLL CS Portion</td>
<td>ID in hexadecimal character format (including leading zeros) Depending on the x ID: S7PMC-specific archive number AR_ID or S7-specific R_ID during BSEND</td>
</tr>
</tbody>
</table>
Example of an archive tag name generated for the S7PMC: #00000001#A#0014

PdeShowDialog

```c
#include <winconmr.h>

BOOL WINAPI PdeShowDialog(
    LPVOID hwnd,
    LPTSTR lpszArcVarName,
    DWORD dwArcVarNameLength,
    LPCSTRING lpVarKey,
    LPCMNR_ERROR lpError,
);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwnd</td>
<td>Window handle</td>
</tr>
<tr>
<td>lpszArcVarName</td>
<td>Pointer pointing to the string field that stores the format DLL-specific archive tag name portion</td>
</tr>
<tr>
<td>dwArcVarNameLength</td>
<td>Maximum length of the format DLL-specific name portion</td>
</tr>
<tr>
<td>lpVarKey</td>
<td>Pointer pointing to the Varkey of the raw data tags</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Function successful</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>

### 5.3.4.4 Online Services

#### Register all Messages

This function is required since the format DLL does not have configuration information about the relevant messages. But messages are not sent by the PLC until the application (WinCC) has logged on to receive messages. Alarm Logging currently calls the MldRegisterMsg function for every relevant message and in this way transfers the configuration information for the single message to the format DLL. Apart from a message description, the format DLL also receives a pointer pointing to the raw data tag (connection) that is assigned to this message. This means that the format DLL can create a table in the main memory during runtime, with which S7PMC-specific logon messages can be structured.

MldRegisterMsg

```c
#include <winccmr.h>

BOOL WINAPI MldRegisterMsg(
    LPDCVARKEY lpDwVarKey,
    LPPDWORD lpMsgNumber,
    LPCMNR_ERROR lpError);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpDMVarKey</td>
<td>Pointer pointing to the Varkey of the raw data tags</td>
</tr>
<tr>
<td>lpMsgNumber</td>
<td>Pointer pointing to the field with single message numbers</td>
</tr>
<tr>
<td>dwNumMsgNumber</td>
<td>Quantity of single message numbers</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
</tbody>
</table>

**Return Description**
- **TRUE**: Function successful
- **FALSE**: Error in the API function, description of the error cause via the pointer lpError

### 5.3.4.5 Registration of all Archive Tags

This function is required because the format DLL does not have the configuration information about the relevant archive tags. The PdeSendMsg function is therefore called for a certain number of relevant archive tags. In this way, the configuration information and additional information of Tag Logging are made known for the archive tags.

Multiple archive tags of a connection can be registered with a single call.

Tag Logging transfers one double word per archive tag as additional information to the format DLL, which is retained in the memory of the format DLL memory. This additional information is required by Tag Logging as soon as the archive tags have to be processed (in the Callback function TagLogging_ARCHIVE_CALLBACK).

This means that the format DLL can create a table in the main memory during runtime, with which the S7PMC-specific logon messages can be structured for the respective archives. The logon messages are required to announce to the PLC the ready to receive status for the respective archive number. Only after the successful logon, does the PLC send the archive data to the application (WinCC).

PdeSendMsg

```c
#include <wincnrm.h>

BOOL WINAPI PdeSendMsg(
    NORM_SENDPROC lpfrCallBack,
    DWORD dwFunctionId,
    LPCTSTR lpArcVarName,
    DWORD dwData,
    LPCTSTR lpVarName, 
    LPVOID lpArchVarName,
    LPDM_VARKEY lpVarKey,
    LPVOID lpUser,
    LPCMN_ERROR lpError
);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpfnCallBack</td>
<td>Pointer pointing to the callback routine with which the raw data tag generated by the format DLL is transferred to the data manager (DM). If zero, the callback routine is called from the INI structure. The function address from the INI structure is not identical with this parameter.</td>
</tr>
<tr>
<td>dwFunctionId</td>
<td>Function identifier FUNC_ID_REGISTER (compare to table below), the same function applies to all listed tags</td>
</tr>
<tr>
<td>lpszArcVarName</td>
<td>Pointer pointing to a pointer field whose elements refer to the archive tag names</td>
</tr>
<tr>
<td>lpdwData</td>
<td>Pointer pointing to a field whose elements contain additional data for the archive tags, can also be zero.</td>
</tr>
<tr>
<td>dwNumArchVarName</td>
<td>Quantity of the archive tag names to be processed</td>
</tr>
<tr>
<td>lpVarKey</td>
<td>Pointer pointing to the Varkey of the raw data tags</td>
</tr>
<tr>
<td>lpUser</td>
<td>Pointer pointing to the user data, transferred unmodified to Callback</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
</tbody>
</table>

Possible functions of the PdeSendMsg procedure (values from dwFunctionId):

<table>
<thead>
<tr>
<th>DEFINE</th>
<th>Bit Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_ID_LOCK</td>
<td>0x00000001</td>
<td>Lock archive tag</td>
</tr>
<tr>
<td>FUNC_ID_FREE</td>
<td>0x00000002</td>
<td>Enable archive tag</td>
</tr>
<tr>
<td>FUNC_ID_REGISTER</td>
<td>0x00000004</td>
<td>Register archive tag</td>
</tr>
<tr>
<td>FUNC_ID_UNREGISTER</td>
<td>0x00000008</td>
<td>Unregister archive tag (currently not required)</td>
</tr>
</tbody>
</table>
5.3.4.6 Language Switch

The configuration dialog must be language-dependent, i.e. the format DLL must recognize the currently set language. During the startup, the language setting is reported in the start structure. The dynamic language switch must also be forwarded to the format DLL by Tag Logging and Alarm Logging. For this purpose, the following call is available:

NormSetLanguage

```c
#include <winccarn.h>

BOOL NormSetLanguage(
    DWORD dwLocaleID,
    LPCMN_ERROR lpError
);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwLocaleID</td>
<td>Language setting at the time of the call</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the default WinCC error structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Function successful</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>
5.3.5 Formatting

If an application has logged on at the PLC to receive messages or archive data, this data will be provided via the respective raw data tag. The format DLL logs on during registration. From that point on, data messages can be sent by the PLC. Data messages are packaged in raw data tags and forwarded to the format DLL through the channel DLL, the data manager and the respective application (in this case, Tag Logging or Alarm Logging); the format DLL is responsible for the raw data tag type. The format DLL interprets the incoming data and constructs messages or archive data from them.

5.3.5.1 Derivation of Single Messages

The content of a raw data tag (of a message) can store n single messages. The format DLL has to interpret this S7PMC-specific message and forward the resulting single messages to Alarm Logging.

The message number (EV_ID) of the S7PMC is a part of the WinCC message number. In a single message, up to ten process values can be delivered by the S7PMC. In this case, the string type is permitted as a process value. This process value type is not supported by Alarm Logging - additional values of such kind have to be rejected by the format DLL.

The MldReceiveMsg function is called by Alarm Logging every time the status of the raw data tag changes, i.e. if the faulty status following OK or vice versa is detected by the data manager. The change of status of the raw data tag (corresponding to a connection) is only of importance for the S7PMC format DLL. Additional information can be found in the chapter Processing in the Event of a Status Change.

MldReceiveMsg

```c
#include <winccnrm.h>

BOOL WINAPI MldReceiveMsg(
    MSG_RECEIVE_MSGPROC lpfnMsgReceive,
    LPDM_VAR_UPDATE_STRUCT lpDMVar,
    LPVOID lpUser,
    LPERROR lpError
);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpfnMsgReceive</td>
<td>Pointer pointing to the callback routine with which the single message structured by the format DLL is transferred to Alarm Logging.</td>
</tr>
<tr>
<td>lpDMVar</td>
<td>Pointer pointing to the raw data tag</td>
</tr>
<tr>
<td>lpUser</td>
<td>Pointer pointing to the user data, transferred unmodified to Callback</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
</tbody>
</table>

**Return**

| TRUE               | Function successful                                                                                                                     |
| FALSE              | Error in the API function, description of the error cause via the pointer lpError                                                       |

The callback function for sending single messages to Alarm Logging is supplied as follows:

```c
typedef BOOL (*MSG_RECEIVE_MSGPROC)(
    LPMSG_RTCREATE_STRUCT lpMsgCreate,
    DWORD dwMsgId,
    LPVOID lpUser,
    LPERROR lpError);
```
5.3.5.2 Acknowledging, Locking/Enabling Messages

The message and alarm concept of WinCC Alarm Logging and S7PMC makes provision for messages being acknowledged depending on their configuration. The acknowledgment information is known to Alarm Logging, but must also be managed in the message acknowledgment storage of the PLC. In order to achieve this, Alarm Logging sends acknowledgment messages to the PLC using the format DLL corresponding to the connection.

On the basis of this input data, the S7PMC format DLL constructs the corresponding S7PMC messages, which are forwarded to the data manager by the Alarm Logging Callback function NORM_SEND_PROC.

The same procedure applies, if a single message is to be locked/reenabled by Alarm Logging, i.e. their generation is to be disabled/enabled at the source in the PLC.

```c
#include <winccfmt.h>

BOOL WINAPI MldSendMsg
(NORM_SEND_PROC lpfnMsgSend,
 LPMSG_SEND_DATA_STRUCT lpSendData,
 DWORD dwNumData,
 LPVOID lpUser,
 LPCMH_ERROR lpError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpfnMsgSend</td>
<td>Pointer pointing to the Alarm Logging callback routine with which the raw data tag - constructed by the format DLL - is transferred for writing to the PLC. The parameters are described in the chapter Query of the Properties of a Format DLL.</td>
</tr>
<tr>
<td>lpSendData</td>
<td>Pointer pointing to the send data, its structure is in more detail below</td>
</tr>
<tr>
<td>dwNumData</td>
<td>Quantity of the individual jobs to be processed</td>
</tr>
<tr>
<td>lpUser</td>
<td>Pointer pointing to the user data, transferred unmodified to Callback</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
<tr>
<td>Return</td>
<td>Description</td>
</tr>
<tr>
<td>TRUE</td>
<td>Function successful</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>
Structure of the Alarm Logging send data (individual job)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD dwVarID</td>
<td>Raw data tag ID of the DM</td>
</tr>
<tr>
<td>DWORD dwNotify</td>
<td>Notify: Possible values</td>
</tr>
<tr>
<td></td>
<td>MSG_STATE_QUIT: Acknowledge message</td>
</tr>
<tr>
<td></td>
<td>MSG_STATE_LOCK: Lock message</td>
</tr>
<tr>
<td></td>
<td>MSG_STATE_UNLOCK: Enable message</td>
</tr>
<tr>
<td></td>
<td>MSG_STATE_QUIT_EMERGENCY: Acknowledge all messages</td>
</tr>
<tr>
<td>DWORD dwData</td>
<td>For QUIT, LOCK, UNLOCK --&gt; message number</td>
</tr>
<tr>
<td></td>
<td>For EMERGENCY ACK --&gt; not used</td>
</tr>
</tbody>
</table>

5.3.5.3 Processing in the Event of a Status Change

The status change of a connection (raw data tags) must be reported to the format DLL. This is carried out by the MIdReceiveMsg function.

<table>
<thead>
<tr>
<th>Status Change from - to</th>
<th>Processing in the S7PMC Format DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faulty - OK</td>
<td>Logon messages for all S7PMC message classes - for which at least one message has been configured - are transferred to the PLC. The logon is performed for S7PMC message class-specific. The format DLL already knows all the configured messages due to their registration.</td>
</tr>
<tr>
<td>OK - Faulty</td>
<td>The format DLL has to reject active jobs which have already been sent to the PLC, but could not be fully processed because of the status change (acknowledgments are missing).</td>
</tr>
</tbody>
</table>

5.3.5.4 Message Update of the S7PMC Format DLL

In the case of a message update, the S7PMC format DLL reads the message status of all the messages that have been reported (via registrations) and sends it as a single message to Alarm Logging. Consequently, it is possible to build on a consistent message picture upon system startup.

A message update is required, if
- a status change from faulty to OK has been detected (that is also the case implicitly upon system startup)
- the PLC sends a message update message. This message is sent to each logged on participant, if, for example, a message overflow has been detected, messages from other participants are acknowledged or enabled.

During the message update, the PLC sends the message acknowledgment states and the lock IDs. The additional values and the time are not sent. In this case, the format DLL supplies the current system time as the time for the single message or writes the ID MSG_STATE_UPDATE into the message status.
5.3.5.5 Formatting of Archive Tags

The format DLL provides two functions to Tag Logging:

- The derivation of individual archive tag values from the content of a raw data tag
- Locking/enabling of archive tags

5.3.5.6 Derivation of individual Archive Tag Values

The content of a raw data tag (of a message) can store n archive tag values. The format DLL has to interpret this S7PMC-specific message and forward the resulting archive tag values to Tag Logging.

For an archive tag, process value converters can also be sent. The S7PMC format DLL will then perform the desired conversion from the process value to the archive tag value. This process involves existing WinCC scaling functions. The exact procedure still has to be specified.

The PdeReceive function is called by Tag Logging every time the status of the raw data tag changes, i.e. if the faulty status following OK or vice versa is detected by the data manager. The change of status of the raw data tag (corresponding to a connection) is only of importance for the S7PMC format DLL. Additional information can be found in the chapter Processing in the Event of a Status Change.

PdeReceive

```c
#include <winccparm.h>

BOOL PdeReceive (LPDM_VAR_UPDATE_STRUCT lpUnVarUpdate,
                 TagLogging_ARCHIVE_CALLBACK lpfnCallback,
                 LPVOID lpUser,
                 LPCMN_ERROR lpError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpUnVarUpdate</td>
<td>Pointer pointing to the raw data tag</td>
</tr>
<tr>
<td>lpfnCallback</td>
<td>Pointer pointing to the callback routine with which the format DLL transfers the individual archive tag values to Tag Logging.</td>
</tr>
<tr>
<td>lpUser</td>
<td>Pointer pointing to the user data, transferred unmodified to Callback</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
</tbody>
</table>

**Return Description**

- TRUE: Function successful
- FALSE: Error in the API function, description of the error cause via the pointer lpError

The callback function for sending individual archive tag values to Tag Logging is supplied as follows:

```c
BOOL (*PDE_ARCHIVE_CALLBACK) (LPCTSTR lpszArcVarName,
                              DOUBLE dwValue,
                              SYSTEMTIME lpTime,
                              DWORD dwFlags,
                              DWORD dwData,
                             ользователем lpUser,
                              LPCMN_ERROR lpError);
```
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszArcVar</td>
<td>Archive tag name as from the raw data ID</td>
</tr>
<tr>
<td>doValue</td>
<td>Archive tag value</td>
</tr>
<tr>
<td>lpstTime</td>
<td>Pointer pointing to the time stamp derived from the user data of the raw data tag</td>
</tr>
<tr>
<td>dwFlags</td>
<td>IDs, whose exact meaning still has to be specified.</td>
</tr>
<tr>
<td>dwData</td>
<td>Additional date that was provided upon registration, is transferred without modification</td>
</tr>
<tr>
<td>lpUser</td>
<td>Pointer pointing to the user data, applied without modification from the function call</td>
</tr>
<tr>
<td>lpError</td>
<td>Pointer pointing to the WinCC error structure</td>
</tr>
</tbody>
</table>

### Return Description

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Function successful</td>
</tr>
<tr>
<td>FALSE</td>
<td>Error in the API function, description of the error cause via the pointer lpError</td>
</tr>
</tbody>
</table>

#### 5.3.5.7 Locking/Enabling of the Archive Tags

With this function, Tag Logging has the option in S7PMC of controlling the receipt of archive tag values. For this purpose, the S7PMC format DLL forms the call for the logoff or logon of the respective archive, and forwards this call to the DM via NORM_SEND_PROC.

From the point of view of the S7PMC format DLL, locking/enabling of archive tags is almost identical to the functions which are required for registering an archive tag. For both functions, the same function is therefore called in the format DLL (PdeSendMsg). Via the dwFunctionId function identifier, the differentiation between registering and locking/enabling is made: for locking/enabling, the additional data lpdwData for each archive tag has no meaning. See also chapter Registration of all Archive Tags.

#### 5.3.5.8 Processing in the Event of a Status Change

The status change of a connection (raw data tags) must be reported to the format DLL. This is carried out by the PdeReceive function.

<table>
<thead>
<tr>
<th>Status Change from - to</th>
<th>Processing in the S7PMC Format DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faulty - OK</td>
<td>Logon messages for all archive tags of all connections.</td>
</tr>
<tr>
<td></td>
<td>The format DLL already knows all the configured messages due to their registration.</td>
</tr>
<tr>
<td>OK - Faulty</td>
<td>The format DLL has to reject active jobs which have already been sent to the PLC, but could not be fully processed because of the status change (acknowledgments are missing).</td>
</tr>
</tbody>
</table>
5.4 Global Library

Global Library
- Displays
  - Displays
  - Meters
  - Scaling
  - Text Fields
  - Windows
- Operation
  - Buttons 3D
  - Buttons Language
  - Controller
  - IncrDecr-Buttons
  - Keyboards
  - Screen Buttons
  - Screen Navigation
  - Sliders
  - Toggle Buttons
- PlantElements
  - Motors
  - PC / PLC
  - Pipes
  - Pipes - Smart Objects
  - Pumps
  - Tanks
  - Valves
  - Valves - Smart Objects
- Symbols
  - Conveyors
  - DIN30600
  - E-Symbols
- ISA Symbols
  - isa_s55a
  - isa_s55b
  - isa_s55c
  - isa_s55d
  - isa_y32a
  - isa_y32b
  - isa_y32c
  - isa_y32d
  - isa_y32e
  - isa_y32f
  - isa_y32g
  - isa_y32h
  - isa_y32i
- Miscellaneous1
  - Miscellaneous2
  - Motors
  - Shut-Off Devices
  - Shut-Off Valves
  - Valves
5.4.1 System Blocks

- PlantElements
  - Motors
  - PC / PLC
  - Pipes
  - Pipes - Smart Objects
  - Pumps
  - Tanks
  - Valves
  - Valves - Smart Objects

5.4.1.1 Motors

- Motor001
- Motor002
- Motor003
- Motor004
- Motor005
- Motor006
- Motor007
5.4.1.2 PC/PLC

5.4.1.3 Pumps

- Pump001
- Pump002
- Pump003
- Pump004
- Pump005
- Pump006
- Pump007
- Pump008
- Pump009
- Pump010
- Pump011
5.4.1.4 Pipes

5.4.1.5 Pipes - Customized Objects
5.4.1.6 Tanks

Tank1  Tank2  Tank3  Tank4

5.4.1.7 Valves - Customized Objects

Valve1  Valve2  Valve3  Valve4

5.4.1.8 Valves

Valve1  Valve2  Valve3  Valve4
5.4.2 Displays

5.4.2.1 Displays

8-Bit Display  8-Bit Display + I/O Field  Digital Output

5.4.2.2 Windows

1  2  3  4

5  6

5.4.2.3 Scaling

01  02  03  04

5.4.2.4 Text Fields

Siemens WinCC  Text  Text: Password Error
5.4.2.5 Meters

Meter1_0-100  Meter1_Min-Max  Meter2_Min-Max  Meter3_Min-Max
5.4.3 Controls

5.4.3.1 3D Buttons

Arrow Down  Arrow Left  Arrow Right  Arrow Up  Double Arrow Down

Double Arrow Left  Double Arrow Right  Double Arrow Up  Magnifying Glass  Matrix 1x3

Matrix 3x3  Parallel Arrows Left  Parallel Arrows Right  Single Arrow Down  Single Arrow Left

Single Arrow Right  Single Arrow Up  Standard Button  Text with Leap

5.4.3.2 Control Panels

Slider1  Slider2  Slider4  Slider5
5.4.3.3 Picture Buttons

5.4.3.4 Picture Navigation

5.4.3.5 Increment/Decrement Buttons
5.4.3.6 Controllers

![Image of a control panel](image)

5.4.3.7 Language Switch

<table>
<thead>
<tr>
<th>Button Eng [US]</th>
<th>Button Fr.</th>
<th>Button Ger.</th>
<th>Hotkey D (German)</th>
<th>Hotkey F (French)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="USA" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hotkey USA (English)</th>
<th>Lang. Switch All</th>
<th>Lang. Switch to Ger or Eng (US)</th>
<th>Lang. Switch to Ger or Fr</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="USA" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.4.3.8 Keyboards

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>Keyboard Char</th>
<th>Keyboard Num</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Keyboard" /></td>
<td><img src="image" alt="Keyboard Char" /></td>
<td><img src="image" alt="Keyboard Num" /></td>
</tr>
</tbody>
</table>
5.4.3.9 Shift Buttons

On_Off_1  On_Off_2  On_Off_3  On_Off_4

On_Off_5  On_Off_6  On_Off_7  On_Off_8
5.4.4 Symbols

5.4.4.1 Shut-Off Devices

[Diagram of Shut-Off Devices]
5.4.4.2 Shut-Off Valves

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
5.4.4.3 DIN 30600

01  02  03  04  05
06  07  08  09  10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
26  27  28  29  30
31  32  33  34  35
5.4.4.4 E Symbols

![Diagram of various symbols](image-url)
5.4.4.5 Conveyors

1  2  3

4  5  6
5.4.4.6 ISA Symbols

5.4.4.6.1 isa_s55a
5.4.4.6.2 isa_s55b

5.4.4.6.3 isa_s55c

5.4.4.6.4 isa_s55d

5.4.4.6.5 isa_y32a
5.4.4.6.6 isa_y32b

5.4.4.6.7 isa_y32c

5.4.4.6.8 isa_y32d

5.4.4.6.9 isa_y32e
5.4.4.6.10  isa_y32f

5.4.4.6.11  isa_y32g

5.4.4.6.12  isa_y32h

5.4.4.6.13  isa_y32i
5.4.4.7 Motors

1_1
1_2
1_4
2_1
2_2
2_4
3_1
3_2
3_4
4_1
4_2
4_4
5_1
5_2
5_4
6_1
6_2
6_4
5.4.4.8 Valves

![Diagram of various valve symbols: 01 to 53]
5.4.4.9 Miscellaneous 1

01 02 03 04 05

06 07 08 09 10

11 12 13 14 15

16 17 18 19 20

21 22 23 24 25

26 27 28 29 30

31 32 33 34 35
5.4.4.10 Miscellaneous 2

01  02  03  04  05
06  07  08  09  10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
26  27  28
Index

A

Access, 3-55, 5-9
  Reading WinCC Data, 5-9
  Rights, 3-61
  To the Database, 3-55, 3-66
Access Protection, 3-15, 3-19, 3-54, 3-77
  Transfer of the Rights, 3-77
Acknowledgment
  Messages, 3-84
Actions, 3-23, 4-12
  At Open Picture, 5-4
  Create, 4-3, 4-14
  Edit, 4-11
  Editor for, 4-11
  Folder, 3-49
  In WinCC, 4-4
  Reuse, 4-11
  Selection, 3-33
  Specify, 3-9
  Transfer, 3-65
  Update Cycles, 3-27
  Update in Picture, 3-23
ActiveX, 2-2, 2-3, 3-57, 3-103
Adaptation, 3-23
  Computer Property, 3-58
  Data for Tag Import, 3-72
  Picture Update, 3-23
Addition, 4-31
Addressing, 3-6
  Indirect, 3-6, 3-96
Alarm, 3-20
  General Information regarding the Specification, 3-20
  In the Control Concept, 3-16
Alpha Cursor, 3-15, 3-83
Analog Values
  Time, 3-103
ANSI, 4-3
API, 2-4, 3-66, 3-67, 3-104
Application, 2-2, 3-40
  Close, 3-52
  Integrate other, 2-3
  Interface (API), 3-66
  Own, 2-2
Application Window, 2-3, 3-60, 3-84, 3-87
Archiving, 3-3, 3-56
  Archiving Times, 3-23
  Configure, 3-59
Area, 3-14
  Key Area, 3-17
  Screen, 3-12
  Time Range, 3-87
  Value Range, 3-42, 4-20
  Workspace, 3-14
Assignment Lists, 3-66
Authorization, 3-57
Autostart, 3-51
  Entry in Folder, 3-51

B

Backup, 3-48, 3-54
  Concept for, 3-54
  WinCC Data, 3-55
Basic, 2-2, 3-59, 3-103
  Basic Project, 3-59, 3-77
  For Import, 3-69
Basic Mathematical Operations, 4-33
Basic Process Control, 3-16, 3-82, 3-87
Bit Message Procedure, 3-20
Bit Operations, 4-36
Bitmap, 3-62
  Transfer, 3-62
Bus System, 3-18

C

Callback Function, 5-65
  For the Transfer of the Process Value read, 5-65
  For the Feedback of the Status of a Write Job, 5-65
C-API, 2-4
Channel, 3-57
  DLL, 3-57, 3-67, 3-72
  S7 PMC, 3-105
Character, 4-42
  String, 4-42
  Valid for Picture Names, 3-7
  Valid for Tag Names, 3-6
  Valid in Project Names, 3-4
Client, 2-4
Color
  Colors in the Project, 3-11
  Definition, 3-17
  Identification, 3-11
Color Table, 5-15
Command, 4-49
  Conditional, 4-49
Communication, 3-18, 3-27, 3-61
  Affect on the Update, 3-27
Between Individual Tasks, 3-28
Interfaces, 3-57
Online Change, 3-105
Process, 3-23
Computer, 3-52
Change Type, 3-104
Folder for, 3-50
Hotkey, 3-85
Settings, 3-52
Concept, 2-5
Control, 3-12
Data Backup, 3-54
Prototype Pictures, 3-94
Conditional, 4-49
Commands, 4-49
Configuration, 3-23, 3-49
Add Dynamics, 3-43
Data from, 3-49
Dialog, 3-23, 3-30, 3-43, 3-57, 3-62, 3-97
Mode, 3-51
Specifications for, 3-2
Configuration Data, 3-49
Connection, 3-18, 3-105
Connection List, 3-71
Indirect, 3-93
Logical, 3-66
New, 3-92
Serial with CP 525, 5-14
Set Up, 3-105
To the Process, 3-18
UPS, 3-54
With Process Tags, 3-101
With Tags, 3-96
Content, 3-49
Display Project Folders, 3-49
Control, 3-12, 3-15, 3-18
Event-Driven, 3-18
Operating Error, 3-10
Trend Windows, 3-87
Via Function Keys, 3-80
Via Keyboard, 3-40, 3-82
Without a Mouse, 3-78
Control Concept, 3-15
Control Objects, 3-83
Conversion, 3-60
Single-User - Multi-User, 3-60
Coros, 3-74
Create, 4-118
File with Script, 4-118
Message Structure, 3-76
Cursor, 3-15
Alpha, 3-15, 3-83
Tab, 3-15, 3-83
Cursor Keys, 3-84, 3-85
Customized Object, 3-63, 3-97, 3-98, 3-101, 3-102
In Picture Module Technology, 3-92
Transfer, 3-63
Wizard, 3-97
Cycle Time, 3-27

D

Data
Backup, 3-56
Configuration Data, 2-4
Data Maintenance, 2-4
Database, 2-5
Import, 3-55
In the Database, 3-47
Request, 3-40
Separation, 3-48
Storage, 3-21
Transfer, 3-59
Transfer from S5 or S7, 3-66
Update, 3-18
Data Export, 5-12
Via C-Action, 5-12
Database, 2-4
Access with ISQL, 5-10
Access with MS Access, 5-9
Access with MS Excel, 5-6
Backup, 3-55
Edit, 3-59
Query Language, 2-4
Reconstruction, 3-47
Selection in, 5-13
DDE, 2-2
Decrement, 4-31
Default, 3-3
Language, 3-46
Settings, 3-3
Trigger, 3-39
Definition, 4-68
C-Structures, 4-68
Development, 2-2
Development Environment, 3-49, 3-103, 4-3
Diagnosis, 3-69, 4-17, 4-18
File for Import, 3-69
Files, 3-45
Scope, 5-5
Dialog, 3-43
Configuration, 3-43, 3-62
Configurations, 3-23
Direct Connection, 3-23, 3-42, 3-82, 3-95, 3-100
Directions, 3-72
For the Data Transfer, 3-72
Division, 4-31
Documentation, 3-2
Drive, 3-55
  For the Data Backup, 3-55
Dynamic, 3-40
  Connection of the Tags, 3-96
  Editing of the Object Properties, 3-40
  Instance, 3-98
Dynamic Dialog, 3-24, 3-28
Dynamic Wizard, 3-23, 3-30, 3-42, 3-66, 3-93, 3-97
Dynamization, 3-97
  Configure, 3-43
  Customized Object, 3-97
  Events, 3-41
  In WinCC, 3-40
  Objects, 3-41
  Properties, 3-40
  Type, 3-28
  Types, 3-23

E
Edit, 3-26
  From States, 3-40
  Picture Cycle, 3-35
  Project Properties, 3-58
  Trigger, 3-33
  User-Defined Cycle, 3-26
  Window Cycle, 3-36
  With Tag Trigger, 3-28
Editor, 4-11
EN 60073, 3-17
End, 3-83
  From WinCC, 3-54
  Input, 3-83
  WinCC, 3-51
Error, 3-48
  Error Message, 3-58
  Find, 3-45, 4-11
  Operating Error, 3-48
Event, 3-41
  Add Dynamics, 3-41
  Controlled, 3-18
  Function Header at, 4-5
  Trigger, 3-23, 3-36, 3-39
Execute, 3-69
  Of Scripts, 3-36
  Tag Import/Export, 3-69
Export, 3-6, 3-47, 3-55, 3-64, 3-69
  Messages, 3-47
  Tags, 3-67
Extension, 3-65
  For Actions, 3-65

WinCC Files, 3-49

F
FIFO Buffer, 3-15
  For Pictures, 3-15
Files, 3-48
  In the Default Folder, 3-45
  In the Project Folder, 3-48
Float, 4-20
Folder, 3-46
  Data in the WinCC Folder, 3-46
  For Autostart, 3-51
  Project Library, 3-65
  Structure of WinCC, 3-21, 3-45
  Tools for WinCC, 3-67
  WinCC Project Folder, 3-48
Font, 3-2
  Color, 3-17
  Size, 3-2, 3-14
  Type, 3-2
  Function Key, 3-78
  Functionality, 3-99
  Generate, 3-58
  Create New Header, 3-58
  New Header, 3-61, 3-77, 3-102
  Graphic, 3-95
  Library, 3-95
  Tools, 3-47
  Group
    Tag Group, 3-5
  Groups, 3-19
    User Group, 3-19, 3-77
**Index**

**H**
- Hardcopy, 3-90
- Hierarchy, 3-15
  - At Plants, 3-80
  - Control, 3-15
  - Picture, 3-82
- HMI, 2-2, 2-5, 3-18, 3-51
- Hotkey, 3-80, 3-81, 3-84, 3-91

**I**
- IF, 4-50
- Import, 3-6, 3-55, 3-66, 3-69
- Increment, 4-31
- Information
  - Find, 3-4
  - In the Picture, 3-10
- Informix, 2-5
- Ingres, 2-5
- Initialization of a Channel-Unit, 5-65
  - Parameters for the WinCC Data Manager, 5-65
- Input
  - Formatted, 5-3
  - In the I/O Field, 3-82
  - Mean, 3-15
- Installation, 3-47
- Tools, 3-47
- WinCC, 3-57
- Instance, 3-98, 3-99
  - Create, 3-99
- Integrate, 3-93
  - From OCX, 3-93
- Interface, 3-66
- API, 3-66
- Internal, 3-27
  - Functions, 3-27, 3-45

**K**
- Keyboard, 3-15
  - Control, 3-15
  - Function Keys, 3-81
  - Hotkey, 3-90
  - Operation, 3-40, 3-78
- Keys
  - Configured, 3-73
  - Function Keys, 3-78
  - Hotkey, 3-81
  - In Message Window, 3-85
  - In Trend Window, 3-87
  - Key Area, 3-12, 3-17
  - Key Assignment, 3-84
  - Key Combination, 3-51
  - Key Settings, 3-83

**L**
- Language, 3-73
  - Transfer, 3-73
- Library, 3-56
  - Default, 3-46
  - Project, 3-56, 3-63, 3-65
  - Text, 3-73
- License Checks, 3-45
- Limitation, 3-60
  - During Data Transfer, 3-60
  - During Online Configuration, 3-104
  - Picture Name, 3-7
  - Tag Name, 3-6
- Log, 3-45
  - Files, 3-45, 3-50
- Logic, 4-32
  - Comparison, 4-32
- Login, 3-91, 3-105
- Logoff, 3-105
- Logout, 3-91
- Loops, 4-49
  - Do while, 4-49
  - For, 4-50
  - While, 4-49

**M**
- Measured Value, 3-18
  - Acquisition, 3-60
  - Archiving, 3-23
  - Export, 3-47
  - Transfer, 3-77
  - Update, 3-18
- Memory, 4-42
- Message, 3-17
  - Message Class, 3-17, 3-74
  - Message List, 3-84
  - Message System, 3-17
- Message Windows, 3-60, 3-84, 3-87
- Messages, 2-4
  - Backup, 3-56
  - Bit Message Procedure, 3-20
  - Message File, 3-75
  - Message Picture, 3-85
  - Message Procedure, 3-20
  - Read In, 3-74
  - Sequenced Reporting, 3-20
Specify Colors, 3-17
System Messages, 3-45
Transfer, 3-73, 3-74
Meter, 3-11, 3-98
Modularity, 2-2
Modules, 2-2, 3-92
Integrate, 2-2
Picture Module Technology, 3-92
Preconfigured, 3-63
Mouse, 3-81
Action for Hotkey, 3-81
Dynamization of Events, 3-41
Operation without a Mouse, 3-78
MS Excel, 3-6, 3-22, 3-69, 3-72, 3-74, 3-76, 5-6
Read Data with MSQuery, 5-6
Multi-Client, 3-60
Multiplication, 4-31
Multi-User, 3-13
System, 3-13, 3-54, 3-60

N
Network, 3-55
Number, 4-20
  Floating-Point Number, 4-20
  Integers, 4-20

O
OCX, 2-3, 3-57, 3-84, 3-93, 3-103
Picture Module, 3-93
Registration, 3-47, 3-103
ODBC, 2-2
OLE, 2-2
  Connections, 3-57
Online, 3-104
  Configuration, 3-104
Operating System, 2-2, 3-18, 3-53, 3-54
Operational
  Control Objects, 3-83
  Functions with Access Protection, 3-54
  Message Windows, 3-84
  Trend Windows, 3-87
Operators, 4-31
Options, 3-16, 3-47, 3-48, 3-82, 3-87
Oracle, 2-5
Overview Pictures, 3-15

P
Parameters, 3-49
  Connection, 3-67
  Default, 3-49
  Diagnosis, 3-105
  For Picture Name, 3-7
  For printf, 4-16
  For Project Name, 3-4
  For Screen Resolution, 3-13
  For Tag Name, 3-6
  Shut Down, 3-54
Password, 3-19, 3-54, 4-11
Performance, 3-2, 3-18, 3-42, 3-59
Picture
  Change, 3-80
  Cycle, 3-26
  Edit Cycle, 3-35
  Hierarchy, 3-82
  Information in the Picture, 3-10
  Message Picture, 3-85
  Module Technology, 3-92
  Names, 3-7
  Object, 3-40
  Reduce Volume, 4-11
  Selection, 3-15, 3-84, 3-87, 3-102
  Size of the, 3-13
  Structure, 3-93
  Transfer of, 3-60
  Update, 3-23
Plant Picture, 3-41
  Add Dynamics, 3-41
  Operation without a Mouse, 3-78
Platform, 2-2
Pointers, 4-41
  In C, 4-41
Print, 4-16
Process, 3-18
  Communication, 3-23
  Connection, 3-18
  Control, 3-12
  Control Hierarchy, 3-15
  Data, 2-2
  Tags, 3-96
Process Data, 2-4, 3-40
Processing, 3-1
Program, 3-51
  Additional Programs, 3-57
  Autostart, 3-51
  For Database, 3-47
  For the Data Transfer, 3-67
  Own, 3-66
  Tools, 3-6
Programming Interface, 2-4
Project, 3-51
  Automatic Startup, 3-51
  Backup, 3-56
  Copy, 3-57
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create Function</strong></td>
<td>4-12, 4-13</td>
</tr>
<tr>
<td><strong>Folder</strong></td>
<td>3-48</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>4-11</td>
</tr>
<tr>
<td><strong>Implementation Tips</strong></td>
<td>3-21</td>
</tr>
<tr>
<td><strong>Library</strong></td>
<td>3-65</td>
</tr>
<tr>
<td><strong>Low-Maintenance</strong></td>
<td>3-27</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>3-4</td>
</tr>
<tr>
<td><strong>Project-Wide Functions</strong></td>
<td>3-46</td>
</tr>
<tr>
<td><strong>Sample Project</strong></td>
<td>3-3</td>
</tr>
<tr>
<td><strong>Transfer Library</strong></td>
<td>3-63</td>
</tr>
<tr>
<td><strong>Properties of a Channel Unit</strong></td>
<td>5-67</td>
</tr>
<tr>
<td><strong>Access to Remote Tag</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Client Functionality</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Diagnostics Options</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Editing the Channel Properties</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Local Cycle Management</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Local Lifebeat Monitoring</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Local Restart Display</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>No Registration of WinCC Tags</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Online Registration of Logical Connections</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Online Registration of WinCC Tags</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Process Values in INTEL Byte Order</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Reentrant</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Time Master</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Time Slave</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Writing to Bit Address</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Writing to Byte Address</strong></td>
<td>5-67, 5-74</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td>3-27</td>
</tr>
<tr>
<td><strong>Add Dynamics at the Object</strong></td>
<td>3-27</td>
</tr>
<tr>
<td><strong>Function Header at the</strong></td>
<td>4-6</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td>3-24</td>
</tr>
<tr>
<td><strong>Prototype</strong></td>
<td>3-92, 3-94, 3-97, 3-99</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>3-41</td>
</tr>
<tr>
<td><strong>Event at Object</strong></td>
<td>3-41</td>
</tr>
<tr>
<td><strong>Keyboard Code</strong></td>
<td>3-78</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Radio-Button</strong></td>
<td>3-84</td>
</tr>
<tr>
<td><strong>Operation via Keyboard</strong></td>
<td>3-84</td>
</tr>
<tr>
<td><strong>Read In</strong></td>
<td>3-74</td>
</tr>
<tr>
<td><strong>Messages</strong></td>
<td>3-74</td>
</tr>
<tr>
<td><strong>Recommendation</strong></td>
<td>3-28</td>
</tr>
<tr>
<td><strong>For Update Cycles</strong></td>
<td>3-28</td>
</tr>
<tr>
<td><strong>Reconstruction</strong></td>
<td>3-47</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>3-47</td>
</tr>
<tr>
<td><strong>Reference</strong></td>
<td>3-95</td>
</tr>
<tr>
<td><strong>In Picture Modules</strong></td>
<td>3-95, 3-102</td>
</tr>
<tr>
<td><strong>In Pictures</strong></td>
<td>3-61</td>
</tr>
<tr>
<td><strong>Text Reference</strong></td>
<td>3-73</td>
</tr>
<tr>
<td><strong>Register</strong></td>
<td>3-47</td>
</tr>
<tr>
<td><strong>OCX</strong></td>
<td>3-47, 3-103</td>
</tr>
<tr>
<td><strong>OLE, OCX</strong></td>
<td>3-57</td>
</tr>
<tr>
<td><strong>Request</strong></td>
<td>3-27</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>3-40</td>
</tr>
<tr>
<td><strong>Data from the Data Manager</strong></td>
<td>3-27</td>
</tr>
<tr>
<td><strong>Requests</strong></td>
<td>3-10</td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
<td>3-13</td>
</tr>
<tr>
<td><strong>Screen</strong></td>
<td>3-13</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>4-16</td>
</tr>
<tr>
<td><strong>Dynamization</strong></td>
<td>3-44</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>4-16</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>4-5</td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td>3-49, 3-97, 3-102</td>
</tr>
<tr>
<td><strong>Control in</strong></td>
<td>3-83</td>
</tr>
<tr>
<td><strong>Cursor</strong></td>
<td>3-83</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>3-49</td>
</tr>
<tr>
<td><strong>Dynamization</strong></td>
<td>3-40</td>
</tr>
<tr>
<td><strong>Dynamization for the</strong></td>
<td>3-40</td>
</tr>
<tr>
<td><strong>Limitation during the Online Configuration</strong></td>
<td>3-104</td>
</tr>
<tr>
<td><strong>Settings for</strong></td>
<td>3-53</td>
</tr>
<tr>
<td><strong>Specifications for</strong></td>
<td>3-2</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Sample Project</strong></td>
<td>3-13, 3-59</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>3-3</td>
</tr>
<tr>
<td><strong>Screen Resolution</strong></td>
<td>3-13</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>5-5</td>
</tr>
<tr>
<td><strong>Screen Resolution</strong></td>
<td>3-13</td>
</tr>
<tr>
<td><strong>Scripts</strong></td>
<td>3-9, 3-36</td>
</tr>
<tr>
<td><strong>Development Environment</strong></td>
<td>4-3</td>
</tr>
<tr>
<td><strong>Editors for</strong></td>
<td>4-11</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>3-36</td>
</tr>
<tr>
<td><strong>For Wizard</strong></td>
<td>3-46</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>4-1</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td>4-3</td>
</tr>
<tr>
<td><strong>Selection</strong></td>
<td>3-15, 3-30, 3-31, 3-33, 3-3</td>
</tr>
<tr>
<td><strong>Messages</strong></td>
<td>3-21</td>
</tr>
<tr>
<td><strong>Of Tags</strong></td>
<td>3-5</td>
</tr>
<tr>
<td><strong>Pictures for Status Display</strong></td>
<td>3-62</td>
</tr>
<tr>
<td><strong>Trigger</strong></td>
<td>3-23</td>
</tr>
<tr>
<td><strong>Sequenced</strong></td>
<td>3-20</td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td>3-20</td>
</tr>
<tr>
<td><strong>Shut Down</strong></td>
<td>3-54</td>
</tr>
<tr>
<td><strong>Single-User</strong></td>
<td>3-13, 3-59</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td>3-13, 3-54, 3-59</td>
</tr>
<tr>
<td><strong>SmartTools</strong></td>
<td>3-6, 3-47, 3-57, 3-67</td>
</tr>
<tr>
<td><strong>Solution</strong></td>
<td>3-51</td>
</tr>
<tr>
<td><strong>Automatic Project Startup</strong></td>
<td>3-51</td>
</tr>
</tbody>
</table>
Solution Approach, 2-2
Special Characters, 3-7
SQL, 2-4, 2-5, 3-52
   Database, 3-52
   Programming, 3-67
Tools, 3-47
Standard, 4-3
   C, 4-3
   Cycles, 3-28
   Functions, 3-56, 4-11
   Keys, 3-82
   Settings, 3-23, 3-25
   WinCC Default Folder Content, 3-45
Start, 3-51
   WinCC Automatically, 3-51
Startup, 3-51
   Automatic, 3-51
   System Messages during, 3-45
Status, 3-20
   Change, 3-20
   Display of States, 3-11
   Status Display, 3-62
String, 4-42
Structure, 3-7, 3-16
   Connection Structure, 3-105
   Connections, 3-101
   Control Concept, 3-16
   Corporate Standard, 3-2
   Create for Messages, 3-76
   Database, 3-59
   Folder Structure, 3-21
   For Data Storage, 3-21
   Functions, 4-5
   Picture Name, 3-7
   Picture Structure, 3-93
   Tag Management, 3-5
   Tags, 3-5
   Text Lists, 3-70
   WinCC, 2-2
   WinCC Project Folder, 3-48
   WinCC System Folder, 3-45
StructurePicture
   Picture Window Structure, 3-100
Structures, 4-68
Subtraction, 4-31
Sybase, 2-4, 3-52
Symbol, 3-46
   Export, 3-64
   For the Dynamization Type, 3-40
   Storage of the, 3-46
   Transfers, 3-62
Syntax, 4-3
System, 3-51
   Automatic Startup, 3-51
   Layout, 3-56
   Load, 3-28, 3-39, 3-40
   Messages, 3-45
   Modules, 2-2
   Multi-User, 3-13, 3-59
   Operating System, 3-18
   Platform, 2-2
   Single-User, 3-13, 3-59
   Software, 3-57
   Tags, 3-6, 3-61
T
   Tab Cursor, 3-15, 3-83
   Table Windows, 3-60, 3-87
   Tags, 3-43
      Archive Tags, 3-89
      Connection, 3-43
      C-Variables, 4-19
      Export, 3-67
      Import/Export, 3-55
      In Scripts, 3-6
      Message regarding missing, 3-45
      Name, 4-20
      Simulation, 3-50
      Specify, 3-5
      Structure of the Exported Lists, 3-70
      Transfer, 3-66
      Transfer from S5/S7, 3-66
      Trigger, 3-23, 3-26, 3-28, 3-39
      Types, 4-20
      Valid Names, 3-6
   Target Group, 4-1
   Task, 3-52
      Formatted, 5-3
      Task Bar, 3-52
      Task Change, 3-27
      Test Output, 4-16
   Tasks, 3-59
      Project Team, 3-59
   Template, 3-3, 3-99
      For Your own Projects, 3-3
      Message Windows, 3-85
      Pictures, 3-99
   Terms, 2-5
   Text, 3-11
      Display, 3-11
      Identification On-Screen, 3-11
      Lists, 3-70
      Multilingual, 3-73
      Text Library, 3-105
      Text Lists, 3-66, 3-84
      Time Cycle, 3-23, 3-27, 3-37
      Time Trigger, 3-36
Index

Toolbar, 3-84
  Alarm Logging, 3-84, 3-87
  Tag Logging, 3-87
Tools, 3-55
  Database, 3-55
  Graphic, 3-47
  Languages, 3-73
  OCX, 3-57
  SQL, 3-47
  Tag Import/Export, 3-6, 3-67
  WinCC, 3-47
Tooltip, 3-14
Transaction-Protected, 2-4
Trends
  Apply Border, 3-60
  Control Windows, 3-85, 3-87
Trigger, 3-23
  In C-Actions, 3-33
  In the Dynamic Dialog, 3-32
  Picture Update, 3-23
  Tag Trigger, 3-28
  Tags, 3-23
  Time-Controlled, 3-37
Turn Off, 3-83
  Runtime Cursor, 3-83

U

UNIX, 2-5
Update, 3-27
  Cycles, 3-18
  Setting Options, 3-23
  Types of, 3-27
UPS, 3-54
User, 2-3, 2-4, 3-10, 3-105
  Authorization, 3-105
  Groups, 3-19, 3-77
  Rights, 3-19
  Transfer of the Rights, 3-77
  User Data Records, 2-4
  User-Defined Time Cycles, 3-25
User Archive, 3-60, 3-77, 3-104
  Transfer, 3-77
User Interface, 2-3, 3-2, 3-10
  Specify, 3-10
User Library, 3-82
User-Defined Cycle, 3-26, 3-37

V

VDE 0199, 3-17

Visual, 2-2
  Basic, 3-103
  C++, 2-2, 3-103
  Visual Basic, 2-2
  Visualization, 3-23
  Station, 3-23
  Visualize, 2-2

W

While, 4-49
WinCC, 3-65
  Actions, 3-9
  Alarming, 3-20
  API, 3-66
  Automatic Startup, 3-51
  Control Concept, 3-15
  Copy Project, 3-57
  Data Backup, 3-55
  Default Folder, 3-45
  Default Setting, 3-3
  Dynamization, 3-40
  End, 3-54
  Folder Structure, 3-48
  Log Files, 3-45
  Project Environment, 3-48
  Scripts, 3-9
  Structure, 2-2, 2-3
  Tag Management, 3-5
  Tools, 3-47, 3-57
  Transfer Action, 3-65
  Transfer Tags, 3-66
  Update Cycle, 3-24
  User Interface, 3-10
  Version 1.10, 3-76
Window, 4-17, 4-18
  Cycle, 3-25, 3-27, 3-36
  Diagnosis, 4-17, 4-18
  Window Change, 3-85
Windows, 2-2, 3-18
  Windows NT, 3-18
Wizard, 3-74
  Files in the Default Folder, 3-46
  For the Picture Update, 3-23
  In Customized Objects, 3-97
  Read in Coros Messages, 3-74
  Read in S5/S7 Tags, 3-66
  Transfer Script, 3-104
Workspace, 3-14
  Wrebuild, 3-47
  Wunload, 3-47