SIEMENS

WinCC

Configuration Manual

Handbuch Band 1

6AV6392-1CA05-0AA0
C79000-G8200-C157-01

Ausgabe September 1999

C79000-G8200-C157
Printed in the Federal Republic of Germany

Siemens Aktiengesellschaft

Inhaltsverzeichnis

1 Projektierungshandbuch 1-1

1.1 Projektierungshandbuch - Hinweise zum Aufbau und zur Verwendung 1-2

2 WinCC - allgemeine Hinweise 2-1

2.1 WinCC - das Konzept 2-2
2.1.1 Die Schnittstellen von WinCC 2-3
2.2 WinCC - Begriffe und ihre Bedeutung 2-5

3 Projektierung - allgemeine Themen 3-1

3.1 Vor Projektbeginn 3-2

3.2 Festlegungen im einzelnen 3-3
3.2.1 Festlegung: WinCC Projektname 3-4
3.2.2 Festlegung: Variablennamen 3-5
3.2.3 Festlegung: Bildnamen 3-7
3.2.4 Festlegung: Scripte und Aktionen 3-8
3.2.5 Festlegung: Die Bedienoberfläche 3-9
3.2.6 Festlegung: Das Bedienkonzept 3-14
3.2.7 Festlegung: Die Farbdefinition 3-16
3.2.8 Festlegung: Die Aktualisierungszyklen 3-17
3.2.9 Festlegung: Die Benutzerrechte 3-18
3.2.10 Festlegung: Die Alarmierung 3-19
3.2.11 Festlegung: Zur Realisierung 3-20

3.3 Besonderheiten bei der Projektierung mit WinCC 3-21
3.3.1 Aktualisierungszyklen - Wie und wo erfolgt die Einstellung 3-22
3.3.1.1 Die Aktualisierung im Bild 3-23
3.3.1.2 Arten der Aktualisierungszyklen 3-24
3.3.1.3 Bedeutung der Aktualisierungszyklen 3-26
3.3.1.4 Hinweise zur Verwendung der Aktualisierungszyklen 3-27
3.3.1.5 Ausführung von Hintergrundscripten (Global Script) 3-34
3.3.2 WinCC Dynamisierung 3-37
3.3.2.1 Dynamisieren der Eigenschaften 3-37
3.3.2.2 Dynamisieren der Ereignisse 3-38
3.3.2.3 Dynamisierungsarten für Objekte 3-39
3.3.3 WinCC Systemumgebung 3-41
3.3.3.1 Verzeichnisstruktur WinCC System 3-41
3.3.4 WinCC Projektumgebung 3-44
3.3.4.1 WinCC Projekt - Verzeichnisstruktur 3-44
3.3.5 Automatischer Projektanlauf von WinCC 3-47
3.3.6 Koordiniertes Beenden von WinCC 3-50
3.3.6.1 Hinweise zur Installation einer USV 3-51
3.3.7 Datensicherung 3-52
3.3.8 Kopieren eines gesicherten WinCC-Projektes auf eine neue Ziel-Maschine 3-54
3.3.9 Wiederverwendung - Übernahme von Projektteilen in ein neues bzw. bestehendes Projekt 3-57

3.3.9.1	Übernahme von Bildern	3-58
3.3.9.2	Übernahme von Symbolen und Bitmaps	3-60
3.3.9.3	Übernahme einer Projektbibliothek (mit vorprojektierten Symbolen und Anwenderobjekten)	3-62
3.3.9.4	Übernahme von Aktionen	3-64
3.3.9.5	Übernahme von Variablen	3-65
3.3.9.6	Übernahme von mehrsprachigen Texten (aus Bildern, in Meldungen)	3-72
3.3.9.7	Übernahme von Meldungen	3-73
3.3.9.8	Übernahme von Meßwerten	3-75
3.3.9.9	Übernahme von Drucklayouts	3-76
3.3.9.10	Übernahme von globalen Aktionen	3-76
3.3.9.11	Übernahme von Projektfunktionen	3-76
3.3.9.12	Nutzen von Standard-Funktionen	3-76
3.3.9.13	Übernahme der Benutzerverwaltung	3-76
3.3.10	Mauslose Bedienung	3-77
3.3.10.1	Bedienung über Tastatur	3-77
3.3.10.2	Bewegung über Bedienobjekte (Eingabefelder und Bedienfelder)	3-82
3.3.10.3	Alarm Logging Funktionstasten zu den Toolbar-Tasten	3-84
3.3.10.4	Tag Logging Funktionstasten zu den Toolbar-Tasten	3-88
3.3.10.5	Druckauftrag anstoßen	3-91
3.3.10.6	An- oder Abmelden	3-92
3.3.11	Bildbausteintechnik	3-93
3.3.11.1	Prozeßbox als Bildbaustein	3-95
3.3.11.2	Bildbaustein mit indirekter Adressierung	3-96
3.3.11.3	Anwenderobjekte	3-98
3.3.11.4	Dynamische Instanz	3-100
3.3.11.5	Prototypbilder	3-101
3.3.11.6	OCX-Objekte	3-105
3.3.12	Online Projektierung (Runtime) - Hinweise, Einschränkungen	3-107

4 WinCC C-Kurs 4-1

4.1	Entwicklungsumgebung für C Scripte	4-3
4.1.1	Aktionseditor des Graphics Designer	4-4
4.1.2	Editor Global Script	4-12
4.2	Variablen	4-19
4.2.1	Beispiel 1 – C Datentypen (Ganzzahlen)	4-21
4.2.2	Beispiel 2 – Definierte Datentypen (Ganzzahlen)	4-23
4.2.3	Beispiel 3 - WinCC Variablen (Ganzzahlen)	4-25
4.2.4	Beispiel 4 – C Datentypen (Gleitkommazahlen)	4-27
4.2.5	Beispiel 5 - WinCC Variablen (Gleitkommazahlen)	4-28
4.2.6	Beispiel 6 – Statische und externe Variablen	4-29
4.3	Operatoren und mathematische Funktionen in C	4-31
4.3.1	Beispiel 1 - Grundrechnungsarten	4-33
4.3.2	Beispiel 2 – Inkrement und Dekrementoperator	4-34
4.3.3	Beispiel 3 - Bitoperationen	4-36
4.3.4	Beispiel 4 – Byteweise Rotieren	4-38
4.3.5	Beispiel 5 – Mathematische Funktionen	4-39

4.4	Zeiger	4-41
4.4.1	Beispiel 1 – Zeiger	4-43
4.4.2	Beispiel 2 – Vektoren	4-44
4.4.3	Beispiel 3 – Zeiger und Vektoren	4-45
4.4.4	Beispiel 4 – Strings	4-47
4.4.5	Beispiel 5 – WinCC Textvariablen	4-48
4.5	Schleifen und bedingte Anweisungen	4-49
4.5.1	Beispiel 1 – while Schleife	4-51
4.5.2	Beispiel 2 – do-while Schleife	4-52
4.5.3	Beispiel 3 – for Schleife	4-53
4.5.4	Beispiel 4 – Endlosschleifen	4-54
4.5.5	Beispiel 5 – if-else Anweisung	4-56
4.5.6	Beispiel 6 – switch-case Anweisung	4-57
4.6	Funktionen	4-58
4.6.1	Beispiel 1 – Übergabe von Wertparametern	4-59
4.6.2	Beispiel 2 – Übergabe von Adressparametern	4-61
4.6.3	Schreiben in übergebenen Adressbereich	4-63
4.6.4	Zurückgeben der Ergebnisadresse	4-65
4.7	Strukturen	4-67
4.7.1	Beispiel 1 – Strukturvariable	4-68
4.7.2	Beispiel 2 – Typdefinition	4-69
4.7.3	Beispiel 3 – WinCC Strukturtyp	4-71
4.7.4	Beispiel 4 – Funktion zum Auslesen eines WinCC Strukturtyps	4-73
4.8	WinCC API	4-76
4.8.1	Beispiel 1 – Eigenschaften ändern über RT Funktion	4-78
4.8.2	Beispiel 2 – Variablenanbindung erstellen über RT Funktion	4-80
4.8.3	Beispiel 3 – Neues Objekt erstellen über CS Funktion	4-82
4.8.4	Beispiel 4 – Eigenschaften ändern über CS Funktion	4-84
4.8.5	Beispiel 5 – Variablenanbindung erstellen über CS Funktion	4-86
4.8.6	Beispiel 6 – Objekte auflisten über CS Funktion	4-88
4.9	Projektumgebung	4-90
4.9.1	Beispiel 1 – Ermitteln der Projektdatei	4-91
4.9.2	Beispiel 2 – Ermitteln des Projektpfades	4-92
4.9.3	Beispiel 3 – Ermitteln des Projektpfades über Projekt-Funktion	4-94
4.9.4	Beispiel 4 – Ermitteln des Installationsverzeichnisses	4-96
4.9.5	Beispiel 5 – Ermitteln des Rechnernamens	4-98
4.9.6	Beispiel 6 – Ermitteln des Benutzernamens	4-99
4.10	Windows API	4-100
4.10.1	Beispiel 1 – Setzen von Fenstereigenschaften	4-101
4.10.2	Beispiel 2 – Auslesen der Systemzeit	4-102
4.10.3	Beispiel 3 – Abspielen von Sounddateien	4-103
4.10.4	Beispiel 4 – Starten eines Programms	4-104
4.11	Standarddialoge	4-105

4.11.1 Beispiel 1 – Sprachumschaltung 4-106
4.11.2 Beispiel 2 – Variablenauswahl 4-108
4.11.3 Beispiel 3 – Fehlerbox 4-110
4.11.4 Beispiel 4 – Fragebox 4-111
4.11.5 Beispiel 5 – Standarddialog Öffnen 4-113

4.12 Dateien 4-115
4.12.1 Beispiel 1 - Daten sichern 4-117
4.12.2 Beispiel 2 - Daten lesen 4-118
4.12.3 Beispiel 3 - Protokollieren 4-119

4.13 Dynamik Wizard 4-121
4.13.1 Erstellung eigener Dynamik Wizard Funktionen 4-122
4.13.2 Struktur einer Dynamik Wizard Funktion 4-123

5 Anhang 5-1
5.1 Tips und Tricks 5-1
5.1.1 Normierte Ein-/Ausgabe am EA-Feld 5-2
5.1.2 Objektspezifische Aktionen bei Bildanwahl 5-3
5.1.3 WinCC-Scope 5-4
5.1.4 Datenbank-Zugriffe 5-5
5.1.4.1 Zugriff auf die Datenbank von Excel/MSQuery aus 5-5
5.1.4.2 Zugriff auf die Datenbank von Access aus 5-8
5.1.4.3 Zugriff auf die Datenbank von ISQL aus 5-10
5.1.4.4 Zugriff auf die Datenbank von WinCC-Scope aus 5-11
5.1.4.5 Export aus Datenbank über C-Aktionen 5-12
5.1.4.6 Datenbank-Selektionen 5-13
5.1.5 Serielle Kopplung 5-14
5.1.6 Farbtabelle 5-15

5.2 Dokumentation des S5-Meldesystems 5-17
5.2.1 Auflistung der Softwarebausteine 5-18
5.2.2 Hardwarevoraussetzung 5-19
5.2.3 Einbinden des S5-Meldesystems in das SIMATIC-S5 Anwenderprogramm 5-20
5.2.3.1 Aufbau des Offset Datenbausteins 5-22
5.2.3.2 Basismeldungsnummer 5-23
5.2.3.3 Offsetmeldungsnummer / Signalzustände der Meldungen 5-24
5.2.3.4 Signalzustandsblock 5-25
5.2.3.5 Adresse des letzten Signalzustandsblocks 5-26
5.2.3.6 Signalzustände 5-26
5.2.3.7 Ruhezustände 5-27
5.2.3.8 Quittierungsbits 5-27
5.2.3.9 Flankenmerker 5-27
5.2.3.10 Aufbau des Parameter Datenbausteins 5-28
5.2.3.11 Aufbau eines Meldungsblocks 5-30
5.2.3.12 Meldungsnummer 5-31
5.2.3.13 Meldungsstatus 5-31
5.2.3.14 Datum-/ Uhrzeit Stempel 5-31
5.2.3.15 Prozeßvariable 5-31

5.2.3.16	Auftragsnummer / Chargenbezeichnung	5-31
5.2.3.17	Reserve	5-32
5.2.3.18	Bildung eines Meldeblocks	5-32
5.2.3.19	Der interne Ringpuffer (FIFO)	5-32
5.2.3.20	Das Sendefach - Datenübertragung zum überlagerten WinCC-System	5-32
5.2.4	Schnittstellenbeschreibung	5-34
5.2.4.1	System Datenbaustein 80	5-34
5.2.4.2	Offset Datenbaustein	5-34
5.2.4.3	Parameter Datenbaustein	5-34
5.2.4.4	Sendefach / Übergabefach	5-35
5.2.5	Parametrierung des S5 Meldesystems / System DB 80	5-36
5.2.6	Projektierungsbeispiel für das S5 Meldesystem	5-42
5.2.6.1	Parametrierung DB 80	5-42
5.2.6.2	Einrichten der Datenbausteine	5-43
5.2.6.3	Initialisierung der Offset Datenbausteine	5-44
5.2.7	Dokumentation der SIMATIC S5 Kommandobausteine	5-47
5.2.7.1	Auflistung der Softwarebausteine	5-48
5.2.7.2	Hardwarevoraussetzung	5-48
5.2.7.3	Aufrufparameter von FB 87: EXECUTE	5-48
5.2.8	Schnittstellenbeschreibung	5-49
5.2.8.1	Projektierungsbeispiel für die S5 Kommandobausteine	5-50
5.2.9	Aufgabe und Funktion der S5-Uhrzeitsynchronisation	5-51
5.2.9.1	Auflistung der Softwarebausteine	5-51
5.2.9.2	Hardwarevoraussetzung	5-51
5.2.10	Aufrufparameter von FB 86 : MELD:UHR	5-52
5.2.11	Datenformate für Datum und Uhrzeit	5-54
5.2.11.1	Uhrendatenbereich CPU 944, CPU 945	5-55
5.2.11.2	Uhrendatenbereich CPU 928B, CPU 948	5-56
5.2.11.3	Uhrendatenbereich CPU 946, CPU 947	5-57
5.2.11.4	Uhrendatenformat für Meldungsblöcke	5-59
5.2.12	Schnittstellenbeschreibung	5-60
5.2.13	Zusammenspiel mit dem WinCC - Meldesystem	5-61
5.3	Schnittstelle Normierungs-DLL zu Alarm Logging und Tag Logging	5-63
5.3.1	Gemeinsame Schnittstelle zu AlarmLogging und TagLogging	5-64
5.3.2	AlarmLogging-spezifische Zusätze	5-65
5.3.3	TagLogging-spezifische Zusätze	5-66
5.3.4	API-Funktionen einer WinCC-Normierungs-DLL	5-67
5.3.4.1	Initialisierung der Normierungs-DLL	5-67
5.3.4.2	Abfrage der Eigenschaften einer Normierungs-DLL	5-67
5.3.4.3	Abfrage des Namens der Normierungs-DLL	5-70
5.3.5	Shutdown der Normierungs-DLL	5-71
5.3.5.1	Erweiterungen der Projektierung	5-71
5.3.5.2	Dialogerweiterung bei der Projektierung von S7PMC-Meldungen	5-71
5.3.5.3	Dialogerweiterung bei der Projektierung von Archivvariablen	5-73
5.3.5.4	Online-Dienste	5-75
5.3.5.5	Registrieren aller Archivvariablen	5-75
5.3.5.6	Sprachumschaltung	5-77
5.3.6	Normierung	5-78

5.3.6.1	Ableitung von Einzelmeldungen	5-78
5.3.6.2	Meldungen quittieren, sperren/freigeben	5-79
5.3.6.3	Bearbeitung beim Zustandswechsel	5-80
5.3.6.4	Meldungsupdate der S7PMC-Normierungs-DLL	5-81
5.3.6.5	Normierung von Archivvariablen	5-81
5.3.6.6	Ableitung von einzelnen Archivvariablen-Werten	5-81
5.3.6.7	Sperren / Freigeben von Archivvariablen	5-83
5.3.6.8	Bearbeitung beim Zustandswechsel	5-83
5.4	Globale Bibliothek	5-85
5.4.1	Anlagenbausteine	5-86
5.4.1.1	Motoren	5-86
5.4.1.2	PC / PLC	5-86
5.4.1.3	Pumpen	5-87
5.4.1.4	Rohre	5-88
5.4.1.5	Rohre – Anwenderobjekte	5-88
5.4.1.6	Tanks	5-89
5.4.1.7	Ventil Anwender Objekte	5-89
5.4.1.8	Ventile	5-89
5.4.2	Anzeigen	5-90
5.4.2.1	Anzeigen	5-90
5.4.2.2	Fenster	5-90
5.4.2.3	Skalierung	5-90
5.4.2.4	Textfelder	5-91
5.4.2.5	Zeiger-Instrumente	5-91
5.4.3	Bedienung	5-92
5.4.3.1	3D-Buttons	5-92
5.4.3.2	Bedienpanels	5-92
5.4.3.3	Bild-Buttons	5-93
5.4.3.4	Bildnavigation	5-93
5.4.3.5	Incr./Decr.-Buttons	5-93
5.4.3.6	Regler	5-94
5.4.3.7	Sprachumschaltung	5-94
5.4.3.8	Tastaturen	5-95
5.4.3.9	Umschalt-Buttons	5-95
5.4.4	Symbole	5-96
5.4.4.1	Absperrarmaturen	5-96
5.4.4.2	Absperrschieber	5-96
5.4.4.3	DIN 30600	5-97
5.4.4.4	E-Symbole	5-98
5.4.4.5	Förderbänder	5-99
5.4.4.6	ISA-Symbole	5-99
5.4.4.7	Motoren	5-102
5.4.4.8	Ventile	5-103
5.4.4.9	Verschiedenes 1	5-104
5.4.4.10	Verschiedenes 2	5-105

Vorwort

Zweck des Handbuchs

Dieses Handbuch stellt Ihnen die Möglichkeiten der Projektierung mit WinCC anhand folgender Teile vor:

- Ein allgemeiner Teil zu WinCC und zu dessen Projektierung
- Eine Einführung in die Scriptverarbeitung
- Ein Anhang

Dieses Handbuch steht sowohl in gedruckter Form sowie als Onlinedokument zur Verfügung.

Über das Inhaltsverzeichniss oder den Index finden sie schnell die benötigten Informationen. Im Onlinedokument steht zusätzlich eine erweiterte Suchfunktionalität zur Verfügung.

Voraussetzungen für dieses Handbuch

Grundkenntnisse in WinCC, zum Beispiel durch das Getting Started, oder praktische Erfahrung in der Projektierung mit WinCC.

Weitere Unterstützung

Bei technischen Fragen wenden Sie sich bitte an Ihren Siemens-Ansprechpartner in der für Sie zuständigen Geschäftsstelle oder Vertretung.

Darüber hinaus steht Ihnen unsere Hotline unter der nachfolgend angeführten Telefonnummer zur Verfügung.

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

Informationen zu SIMATIC Produkten

Ständig aktuelle Informationen zu den SIMATIC Produkten erhalten Sie im Katalog CA01. Dieser kann unter der nachfolgend angeführten Internetadresse erreicht werden.

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

Darüber hinaus bietet Ihnen der SIEMENS Customer Support Unterstützung durch aktuelle Informationen und Downloads. Eine Sammlung häufig gestellter Fragen finden Sie unter der nachfolgend angeführten Internetadresse.

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

1 Projektierungshandbuch

Das Projektierungshandbuch ist Bestandteil der WinCC Dokumentation und befaßt sich hauptsächlich mit dem praktischen Einsatz von WinCC in Projekten.

Einleitung

In den letzten Jahren sind die Anforderungen an ein System zur Beobachtung und Steuerung von Produktionsvorgängen, sowie für die Archivierung und Weiterverarbeitung der Produktionsdaten, stark gestiegen. Um diesen neuen Anforderungen gerecht zu werden, sind in den letzten Jahren neue HMI-Systeme entwickelt worden.

Eines dieser neuen Systeme ist WinCC. Was die Funktionalität, die Offenheit und die Aktualität (Stand der Softwaretechnik) betrifft, ist WinCC sicher einzigartig.

HMI-Systeme der älteren Generation boten zur Lösung der gestellten Aufgaben oft nur einen Weg. Mit WinCC gibt es für die Realisierung dieser Aufgaben fast immer mehrere Möglichkeiten. Damit Sie stets den besten Lösungsweg im Sinne von Performance und Projektierungsaufwand anwenden, wurde dieses Projektierungshandbuch geschrieben.

Diese Beschreibung soll Ihnen Lösungsvorschläge für den effektiven Einsatz von WinCC in Anlagenprojekten zeigen.

Wir haben die Lösungsvorschläge in WinCC Beispielprojekten realisiert. Diese Beispielprojekte werden mit der WinCC CD geliefert. Sie können diese Lösungsvorschläge direkt in Ihren Projekten verwenden und sparen somit wertvolle Zeit.

1.1 Projektierungshandbuch - Hinweise zum Aufbau und zur Verwendung

Voraussetzung

Vor dem Arbeiten mit diesem Projektierungshandbuch sollten Sie bereits praktische Erfahrungen mit WinCC haben. Für WinCC Neueinsteiger ist das Getting Started ein idealer Einstieg. Im Getting Started werden die wichtigsten Themen erklärt und anhand eines kleinen Beispieles praktisch veranschaulicht. Dieses Projektierungshandbuch ist eine Ergänzung zum WinCC Hilfesystem (Online und Dokumentation). Besonderheiten von Objekten, Eigenschaften und anderen Themen sind, soweit nicht in diesem Projektierungshandbuch erklärt, dort nachzulesen.

Inhalt und Aufbau

Die Beschreibung ist in die nachfolgend aufgelisteten Kapitel gegliedert.

- WinCC – allgemeine Hinweise
  Der erste Teil enthält allgemeine Hinweise zum System WinCC.
- Projektierung – allgemeine Themen
  Der zweite Teil beinhaltet allgemeine und spezielle Hinweise zur strukturierten und effektiven Abwicklung von HMI-Projekten.
- WinCC C-Kurs
  Der dritte Teil beinhaltet einen WinCC C-Kurs. Für Neueinsteiger sind darin die wichtigsten Regeln für den Einsatz der WinCC Scriptsprache enthalten. Der C-Profi findet darin die Besonderheiten der Entwicklungsumgebung.
- Anhang
  Der vierte Teil enthält verschiedene weitere Themen.

Konventionen

Im Kommunikationshandbuch werden folgende Konventionen verwendet:

Konvention	Beschreibung
	Kennzeichnet eine Bedienung mit der linken Maustaste.
 **R**	Kennzeichnet eine Bedienung mit der rechten Maustaste.
 **D**	Kennzeichnet eine Bedienung über Doppelklick mit der linken Maustaste.
Kursiv	Kennzeichnet Begriffe aus dem WinCC Umfeld sowie Begriffe, welche auf Elemente der Programmoberfläche verweisen.
Kursiv Grün	Kennzeichnet eine vom Benutzer nachzuvollziehende Bedienreihenfolge oder Eingaben (farbliche Kennzeichnung nur im Onlinedokument).
Blau	Querverweise sind Blau gekennzeichnet (farbliche Kennzeichnung nur im Onlinedokument).

Finden von Informationen

In der gedruckten Version des Kommunikationshandbuches können Informationen auf die nachfolgend beschriebenen Arten gefunden werden.

- Im Inhaltsverzeichnis finden Sie die Informationen nach Themen gegliedert.
- Im Index finden Sie die Informationen nach Stichworten gegliedert.

Im Onlinedokument können Informationen auf die nachfolgend beschriebenen Arten gefunden werden.

- Auf der Registerkarte Inhalt finden Sie die Informationen nach Themen gegliedert.
- Auf der Registerkarte Index können Sie Wörter in einem Stichworteverzeichnis suchen.
- Auf der Registerkarte Suchen können Sie Wörter im gesamten Dokument suchen.

Die im Rahmen dieses Handbuchs beschriebenen Beispielprojekte können direkt aus dem Onlinedokument auf die Festplatte kopiert werden.

2 WinCC - allgemeine Hinweise

Der Aufbau, das Konzept und die Arbeitsweise von WinCC wird Ihnen in diesem Abschnitt näher erklärt.

2.1 WinCC - das Konzept

In WinCC gibt es aus Sicht der Projektierung grundsätzlich drei Lösungsansätze:

- die Projektierung mit WinCC Standardmitteln,
- die Verwendung vorhandener Windows Applikationen mit WinCC über DDE, OLE, ODBC und ActiveX,
- die Entwicklung eigener Applikationen (in VisualC++ oder Visual Basic) eingebettet in WinCC.

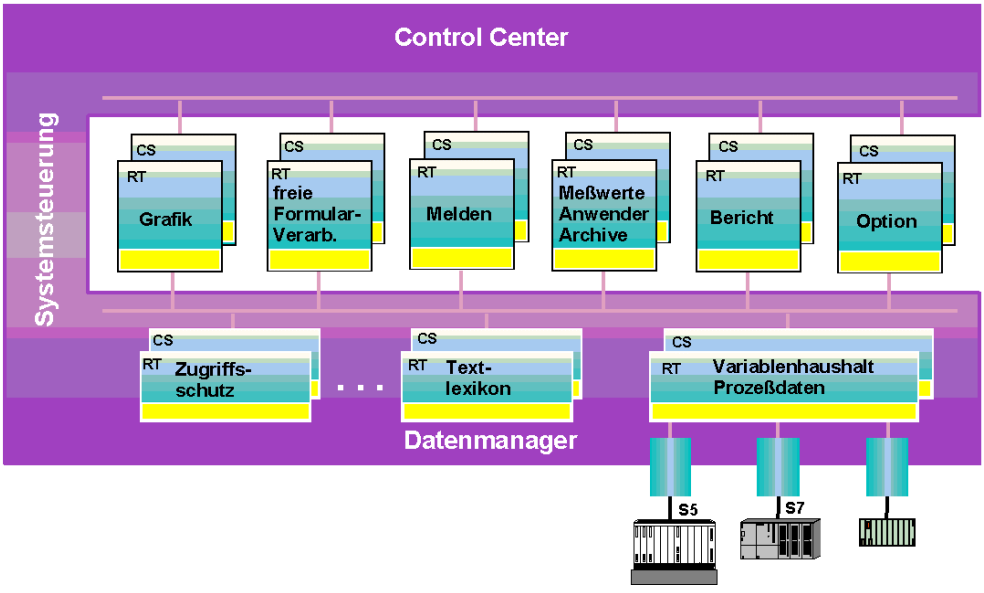
Für die Einen ist WinCC das HMI-System für die kostengünstige und schnelle Projektierung und für die Anderen eine unbegrenzt ausbaufähige Systemplattform. Durch die Modularität und Flexibilität von WinCC bieten sich bei der Planung und Realisierung von Automatisierungsaufgaben völlig neue Möglichkeiten.

Das Betriebssystem: Basis von WinCC

WinCC basiert auf den 32-Bit Betriebssystemen von Microsoft (Windows NT 4.0). Diese Betriebssystem ist das Standard- Betriebssystem auf der PC-Plattform.

Der modulare Aufbau von WinCC

WinCC bietet Systemmodule zum Visualisieren, Melden, Erfassen, Archivieren und Protokollieren von Prozeßdaten, sowie zur koordinierten Einbindung von frei formulierten Anwenderroutinen.
Dabei können auch eigene Module eingebunden werden.

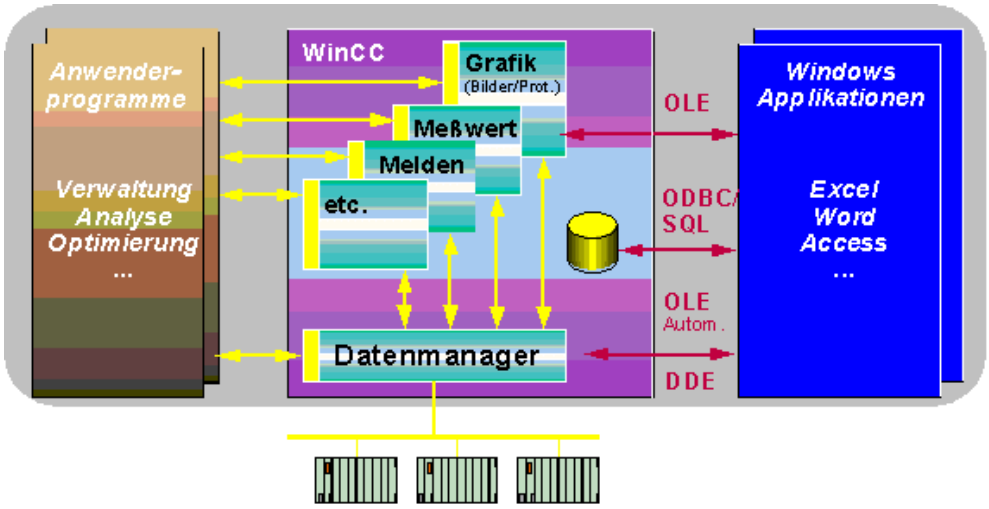


2.1.1 Die Schnittstellen von WinCC

Die Offenheit von WinCC

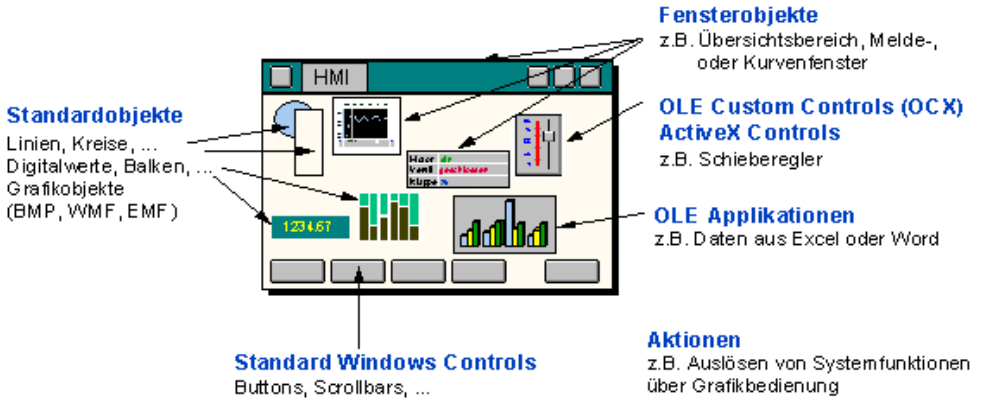
WinCC ist absolut offen für jede Erweiterung durch den Anwender. Diese absolute Offenheit wird durch den modularen Aufbau von WinCC und seiner leistungsfähigen Programmierschnittstelle erreicht.

Das nachfolgende Bild zeigt die Möglichkeiten der Anbindung diverser Anwendungen.



Einbinden fremder Applikationen in WinCC

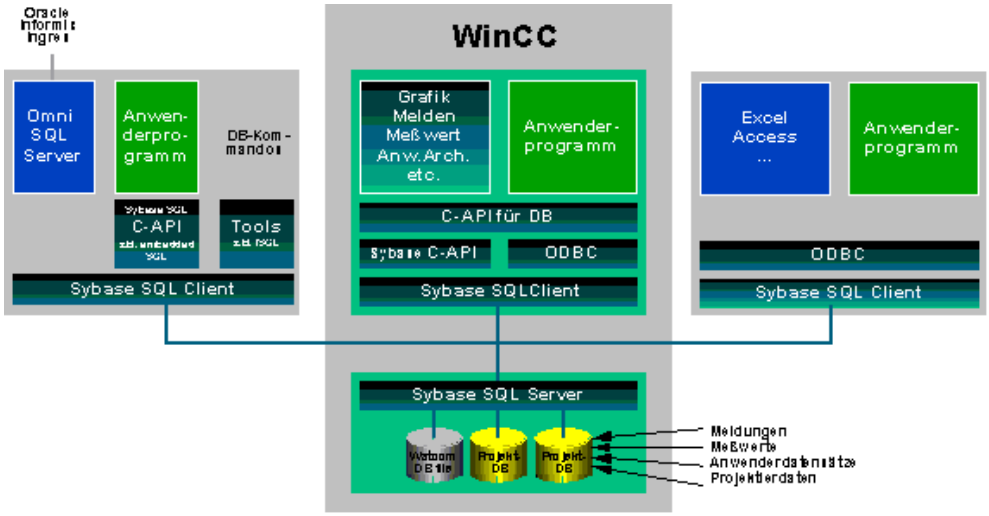
Ganz entscheidend ist, daß WinCC Möglichkeiten bietet, andere Applikationen und Applikationsbausteine **homogen in die Bedienoberfläche für den Prozeßbetrieb zu integrieren.**
Wie nachfolgend zu erkennen ist, können sowohl OLE Applikationsfenster als auch OLE Custom Controls (32-Bit-OCX-Objekte) bzw. ActiveX Controls in die WinCC-Applikation integriert werden, als wären es WinCC-eigene Objekte.



Die Datenhaltung bei WinCC

Im folgenden Schaubild bildet WinCC den gesamten Mittelteil. Die Grafik zeigt, daß die **Standard-Datenbank Sybase SQL Anywhere** WinCC unterlagert ist. Diese wird benutzt, um (transaktionsgesichert) alle listenorientierten Projektierungsdaten wie Variablenlisten und Meldetexte, aber auch aktuelle Prozeßdaten wie Meldungen, Meßwerte und Anwenderdatensätze abzulegen. Diese Datenbank hat Server-Funktion. WinCC kann über ODBC, aber auch über die offengelegte Programmierschnittstelle (C-API) als Client auf die Datenbank zugreifen.

Das gleiche Recht wird natürlich auch anderen Programmen zugestanden. Aus diesem Grunde hat eine **Windows-Tabellenkalkulation** oder eine **Windows-Datenbank** direkten Zugriff auf den Datenbestand der WinCC-Datenbank, unabhängig davon, ob die Anwendung auf dem selben Rechner oder auf einer vernetzten Station ausgeführt wird. Mit Hilfe der Datenbankabfragesprache SQL und entsprechender Connectivity Tools (z.B. ODBC-Treiber) verfügen aber auch andere Clients (z.B. **UNIX-Datenbanken**, wie Oracle, Informix, Ingres) über eine Zugriffsmöglichkeit auf die WinCC Datenbestände. Der umgekehrte Fall gilt natürlich auch. Insgesamt steht damit einer **Einbindung von WinCC in ein werks- oder unternehmensweites Konzept** nichts im Wege.



2.2 WinCC - Begriffe und ihre Bedeutung

Dieser Abschnitt enthält eine Begriffssammlung rund um das WinCC in alphabetischer Reihenfolge. Viele dieser hier beschriebenen Begriffe sind Ihnen wahrscheinlich bereits bekannt:

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

3 Projektierung - allgemeine Themen

In diesem Abschnitt finden Sie viele Hinweise und Ideen für die Abwicklung von Projekten mit WinCC. Einige dieser Hinweise sind nicht WinCC spezifisch.
Im Idealfall sollten diese Festlegungen (Regeln für die Projektierung) die Qualität eines Styleguides für die Projektierung und das Design der Runtime Projekte haben.

3.1 Vor Projektbeginn

Bevor Sie mit der Projektierung beginnen, sollten Sie einige Festlegungen und Strukturierungen vornehmen. Dadurch

- vereinfacht sich die Projektierung,
- erhöht sich die Übersichtlichkeit des Projekts,
- vereinfacht sich die Arbeit im Team,
- verbessert sich mitunter die Stabilität und Performance und
- wird die Wartung der Projekte vereinfacht.

Für den Auf- oder Ausbau eines Firmenstandards ist eine klare Festlegung der Aufbaurichtlinien eine Grundvoraussetzung. Dabei lassen sich Festlegungen für die Projektierung und Festlegungen für das Runtime Projekt unterscheiden.

Festlegungen für die Projektierung

Vor Beginn der Projektierung sollten folgende Festlegungen getroffen werden:

- den Namen für das WinCC-Projekt festlegen.
- die Namen der Variablen festlegen.
- die Namen der WinCC Bilder festlegen.
- Regeln für die Erstellung der Scripte und Aktionen festlegen.
- Regeln für die Projektierung (Firmenstandards, Bibliotheksfunktion, Arbeiten im Team) festlegen.
- Festlegungen für die Dokumentation des Projekts treffen.

Festlegungen für das Runtime Projekt

Festlegungen, die das Runtime Projekt (Ergebnis der Projektierung) betreffen. Diese Festlegungen sind stark vom Einsatzgebiet abhängig (Automobilindustrie, Chemie, Maschinenhersteller, u.a.m.). Es sollten folgende Festlegungen getroffen werden.:

- die Bedienoberfläche (Einteilung des Bildschirms, Schriftart und -Größe, Sprache im Runtime, Darstellung der Objekte) festlegen.
- das Bedienkonzept (Bildhierarchie, Bedienphilosophie, Benutzerrechte, zugelassene Tasten) festlegen.
- die Farbdefinition für Meldungen, Grenzwerte, Zustände, Schrift, u.a.m. festlegen.
- die Kommunikation (Art der Kopplung, Art und Zyklen der Aktualisierung) festlegen.
- das Mengengerüst (Anzahl der Alarme, Archivwerte, Kurven, Clients, u.a.m.) festlegen.
- Melde- und Archivierungsverfahren festlegen.

3.2 Festlegungen im einzelnen

In diesem Teil des Handbuchs treffen wir Festlegungen, die wir in unseren Beispielprojekten verwenden. Diese Festlegungen sollen Ihnen bei der Erstellung eigener Projekte als Vorlage dienen.

Hinweis:
In unseren Beispielprojekten sind die Namen für Projekte, Bilder, Variablen und Kommentare in den Scripten in englischer Sprache.

Defaultwerte der Projektierwerkzeuge

Bei den meisten Editoren in WinCC können bestimmte Eigenschaften per Defaultwert eingestellt werden. WinCC unterstützt somit Ihren Projektierstil und läßt sich so für bestimmte Aufgabengebiete optimal konfigurieren.

Hinweis:
Als Beispiel sind hier die Möglichkeiten im *Graphics Designer* → *Extras* → *Einstellungen...* genannt. Eine ausführliche Beschreibung zu diesem Thema finden Sie in der Hilfe zum *Graphics Designer*.

3.2.1 Festlegung: WinCC Projektname

Allgemeines

Der Name des WinCC Projekts wird defaultmäßig auch für das Verzeichnis vorgeschlagen, in dem alle Daten abgelegt werden. Der Verzeichnisname kann bei Projekterstellung oder zu einem späteren Zeitpunkt (im Windows Explorer) geändert werden.

Parameter / Grenzen

Es sind alle Zeichen außer gewisse Sonderzeichen (z.B. \ ? ' . ; : /) erlaubt. Weiters sind numerische Werte von 0-9 erlaubt.

Festlegung

In den Beispielprojekten, welche im zweiten Teil des Projektierungshandbuchs beschrieben werden, gilt für den Projektnamen folgende Festlegung:

a...a_nn

Die Bedeutung der Kürzel im einzelnen:

a Typbezeichnung (a-z, A-Z, keine Sonderzeichen).
_n Laufende Nummer zur Unterscheidung mehrerer Projekte eines Typs (Zahlen 0-9), Bereich 00-99.

Beispiel: cours_00.mcp, oder pictu_01.mcp

Hinweis für die allgemeine Verwendung

Der WinCC Projektname kann z.B. für die Unterscheidung einzelner Anlagenteile verwendet werden.

Hinweis:
Bei der Rückdokumentation können Sie den Projektnamen in den Ausdrucken mit ausgeben. Die Zuordnung und Auffindbarkeit von Informationen wird dadurch vereinfacht.

3.2.2 Festlegung: Variablennamen

Allgemeines

Die Vergabe von Variablennamen ist nicht mehr auf 8 Zeichen beschränkt. Trotzdem sollte man seine Variablennamen nicht zu lang machen. Die Einhaltung fester Regeln bei der Vergabe der Variablennamen bringt während der Projektierung enorme Vorteile.
Bei der Erstellung von WinCC Projekten ist der Aufbau des Variablenhaushalts eine der Schlüsselaufgaben für die schnelle und effektive Projektierung und performante Verarbeitung im Runtime (in Scripten).

Vor der Festlegung der Variablennamen sind einige Besonderheiten der Struktur des Variablenhaushalts in WinCC zu berücksichtigen. Die Erstellung von Gruppen hat nur Auswirkung auf die Darstellung der Variablen im Variablenhaushalt während der Projektierung. Gruppennamen haben keinen Einfluß auf die Eindeutigkeit der Variablennamen. Die Variablennamen innerhalb eines WinCC Projekts müssen eindeutig sein. Die Eindeutigkeit wird vom System überprüft.

Bei der Auswahl von Variablen bietet WinCC viele Hilfen, z.B. bei der Sortierung nach Spalten (Namen, Erstellungsdatum, u.a.m.), oder bei der Verwendung von Filtern. Es kann aber trotzdem sinnvoll sein, wenn der Variablennamen zusätzliche Informationen enthält.

Festlegung

Für Variablennamen in den Beispielprojekten zu diesem Handbuch gilt:

xxxy_z...z_a...a_nn

Die Bedeutung der Kürzel im einzelnen:

x	Kürzel	Typ
	BIN	Binäre Variable
	U08	Vorzeichenloser 8-Bit Wert *(unsigned)*
	S08	Vorzeichenbehafteter 8-Bit Wert *(signed)*
	U16	Vorzeichenloser 16-Bit Wert
	S16	Vorzeichenbehafteter 16-Bit Wert
	U32	Vorzeichenloser 32-Bit Wert
	S32	Vorzeichenbehafteter 32-Bit Wert
	G32	Gleitkommazahl 32-Bit IEEE 754
	G64	Gleitkommazahl 64-Bit IEEE 754
	T08	Textvariable 8-Bit Zeichensatz
	T16	Textvariable 16-Bit Zeichensatz
	ROH	Rohdatentyp
	TER	Textreferenz
	STU	Strukturtypen

y	Kürzel	Herkunft
	r	reine Lesevariable aus dem PLC (read)
	w	Schreib- und Lesevariable aus dem PLC (write)
	i	interne Variable in WinCC, ohne Verbindung zum PLC
	x	Variable mit indirekter Adressierung (eine Textvariable, deren Inhalt ein Variablenname ist)

_z Gruppe (entspricht Teilanlagen oder Gebäuden)
_Lack ... z.B. Name der Teilanlage

_a Variablenname (z.B. Meßstellennamen)
_EU0815V10 ... z.B. Bezeichnung der Meßstelle

_n Laufende Nummer der Instanz (Zahlen 0-9) Bereich 00-99.

Parameter / Grenzen

Für die Vergabe von Variablennamen bestehen folgende Einschränkungen:

- Das Sonderzeichen @ sollte WinCC Systemvariablen vorbehalten bleiben, die Verwendung dieses Zeichens ist jedoch grundsätzlich möglich.
- Die Sonderzeichen ' und % können nicht verwendet werden.
- Das Sonderzeichen " und die Zeichenfolge // sollen nicht verwendet werden, da diese eine besondere Bedeutung in C-Scripten haben (Einleitung bzw. Abschluß einer Zeichenfolge sowie Einleitung eines Kommentars).
- keine Leerzeichen.
- bei Variablennamen wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Hinweis für die allgemeine Verwendung

Die Vergabe der Variablennamen in den Beispielen soll nur als Vorschlag zu verstehen sein.
Für die Verwendung der Variablen in Scripten und Excel kann es sinnvoll sein, die einzelnen Teile des Variablennamens mit fester Länge (bei Bedarf mit Füllzeichen, 0 oder x) zu vergeben.
Es lassen sich z.B. in Excel große Mengen an Variablen sehr effektiv und einfach erstellen und pflegen. Eine feste Struktur beim Aufbau des Variablennamens erleichtert die Erstellung der Variablenlisten in Excel. Diese in Excel erstellten Variablenlisten können Sie anschließend mit dem Programm *Var_exim.exe*, welches sich auf Ihrer WinCC CD befindet, in das aktuelle WinCC Projekt importieren.

3.2.3 Festlegung: Bildnamen

Allgemeines

Sollten Bilder in Scripten oder externen Programmen angesprochen werden, ist es sehr hilfreich, bei der Vergabe der Bildnamen eine feste Struktur zu verwenden. Die Länge der Bildnamen sollte man sich ebenfalls genau überlegen. Zu lange Namen (Dateinamen) schränken die Übersichtlichkeit eher ein (Auswahl in Listboxen, Aufrufe in Scripten u.a.m.). In der Praxis hat sich eine Länge von maximal 40 Zeichen bewährt.

Parameter/ Grenzen

Für die Vergabe von Bildnamen bestehen folgende Einschränkungen:

- maximale Länge von 255 Zeichen.
- beliebige Zeichen außer gewissen Sonderzeichen (z:B. / " \ : ? < >).:
- bei Bildnamen wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Festlegung

Für Bildnamen in den Projekten zu diesem Handbuch gilt:

aaaaa_k_x...x_nn

Die Bedeutung der Kürzel im Einzelnen:

a Bildkennung (a-z, A-Z, keine Sonderzeichen) für die Gruppierung der Bilder.
course... z.B. Name der Bilder im C-Kurs

_k

Kennung des Bildtyps 0 - 99	Bildtyp
_0	Startbild
_1	Übersichtsbild
_2	Tastenbild
_3	Anlagenbild
_4	Detailbild
_5	Meldebild
_6	Kurvenbild
_7	...
_8	...
_9	Diagnosebilder (nur für Test oder Inbetriebnahme)

_x Name zur Beschreibung der Bildfunktion (a-z, A-Z, keine Sonderzeichen) maximal 30 Zeichen lang.
_chapter ... z.B. Name der Kapitel im C-Kurs

_n Laufende Nummer des Typs (Zahlen 0-9) Bereich 0-99.

3.2.4 Festlegung: Scripte und Aktionen

Allgemeines

Sie können in WinCC Projekten eigene Scripte und Aktionen anlegen. Bei der Vergabe der Namen sollten Sie **aussagekräftige Namen** vergeben. Dies ist bei der späteren Verwendung der Scripte sehr hilfreich.

Bei der Projektierung im Global Script (Editor) ist die Verwendung einer Proportionalschriftart eher störend. Sie sollten deshalb zur besseren Lesbarkeit einen Schriftsatz mit konstanter Zeichenbreite verwenden (z.B. Courier).

Die Scripte sollten immer ausreichend kommentiert werden. Der zeitliche Aufwand für das Erstellen der Kommentare steht in keinem Verhältnis zu der Zeit, die man benötigt, um ein schlecht kommentiertes Programm nachzuvollziehen. Obwohl diese Tatsache hinlänglich bekannt ist, wird sie häufig ignoriert.

Festlegung

Für Scripte in den Projekten zu diesem Handbuch gilt:
Wir verwenden die Proportionalschrift *Courier New* in Größe 8,
alle Variablennamen und Kommentare sind in englischer Sprache.

Hinweis für die allgemeine Verwendung

Eine ausführliche Beschreibung für die Verwendung der Scripte, Aktionen und der Editoren finden Sie im Kapitel 4.1 Entwicklungsumgebung für C Scripte.

3.2.5 Festlegung: Die Bedienoberfläche

Allgemeines

Es ist von größter Bedeutung, daß bei der Erstellung der Bedienoberfläche mit äußerster Sorgfalt vorgegangen wird. Alle Objekte, die im *Graphics Designer* erstellt werden, erscheinen auf dem Bildschirm im Arbeitsraum des Bedieners.
Die damit erstellten Bilder sind die einzige Schnittstelle zwischen Maschine und Anwender und bedürfen damit besonderer Sorgfalt bei der Erstellung, da sie maßgeblich den Erfolg eines Projekts mitbestimmen. Selbstverständlich ist die Funktion der Anlage wichtiger als das Aussehen des Bildschirms, aber auf lange Sicht können nachlässig erstellte Bilder den Eindruck einer ansonsten gut durchdachten Anlage negativ beeinträchtigen bzw. die Wartungskosten erhöhen.
Diese Bilder sind diejenigen, welche der Anwender (Kunde) täglich sieht.

In einem Bildschirmsystem werden dem Anwender einzig und allein über die Bilder Informationen über den jeweiligen Status der Anlage gegeben. Diese Schnittstelle muß also **möglichst umfassende und leicht verständliche Informationen** zur Verfügung stellen.
In WinCC können Sie die Bedienoberfläche frei gestalten. Wie Sie die Bedienoberfläche des Systems gestalten, hängt von der eingesetzten Hardware, von den Anforderungen im Prozeßbetrieb und von bereits vorhandenen Festlegungen ab.

Der Bediener

Bei der Festlegung der Bedienoberfläche sollten Sie den Bediener, für den die Projektierung letzten Endes ausgeführt wird, in den Mittelpunkt Ihrer Betrachtung stellen.

Wenn es gelingt, dem Bediener die erforderlichen Informationen zu geben und diese auf eine übersichtliche Weise darzustellen, wird das Ergebnis eine **höhere Qualität der Produktion und weniger Betriebsunterbrechungen** sein. Die Wartungsarbeiten werden verringert.

Der Bediener braucht so viel Informationen wie möglich. Auf der Grundlage von diesen Daten kann er Entscheidungen treffen, die erforderlich sind, um den Prozeß mit hoher Qualität in Gang zu halten. Der Bediener soll nicht in erster Linie auf Alarme reagieren (dann ist der Prozeß schon aus dem Gleichgewicht geraten), sondern er soll mit Hilfe seiner Erfahrung, seiner Prozeßkenntnisse und der Informationen vom Bediensystem vorhersagen können, in welche Richtung sich der Prozeß entwickelt. Der Bediener soll einer Unregelmäßigkeit entgegenwirken können, bevor sie entsteht. Mit WinCC haben Sie die Möglichkeit, dem Bediener diese Informationen effektiv aufzubereiten und anzuzeigen.

Wieviele Informationen sollen in ein Bild?

Für die Entscheidung wieviel Informationen in ein Bild sollten, gibt es zwei Aspekte, die in einem ausgewogenen Verhältnis stehen müssen:

- sind in einem Bild zu viele Informationen enthalten, wird es schwer lesbar und die Suche nach Informationen wird zuviel Zeit erfordern. Die Wahrscheinlichkeit einer Fehlbedienung wird erhöht.
- sind in einem Bild zu wenig Informationen enthalten, wird die Arbeitsbelastung für den Bediener erhöht. Er verliert den Überblick über den Prozeß. Er muß häufig das Bild wechseln, um die benötigte Information zu finden. Verspätete Reaktionen, Bedienungen und Instabilität des zu kontrollierenden Prozesses sind die Folge.

Untersuchungen zeigen, daß ein erfahrener Bediener **so viele Informationen wie möglich in jedem Bild** wünscht, um das Bild nicht so oft wechseln zu müssen.
Ein Anfänger dagegen wird bei vielen Informationen in einem Bild verwirrt und verunsichert. Er findet die richtige Information nicht oder nicht rechtzeitig.
Aber aus der Erfahrung wissen wir: **Ein Anfänger wird bald erfahren, aber ein Erfahrener wird nie wieder unerfahren**.

Informationen ausblenden

Die Informationen, die gezeigt werden, sollen wichtig und leicht verständlich sein. Sie können bestimmte Informationen solange ausblenden (z.B. Meßstellenkennzeichen), bis sie benötigt werden.

Darstellung der Informationen

Kombinieren Sie bei der Darstellung von Analogwerten Zeigerinstrumente mit Digitalwerten. Für eine schnelle Informationsaufnahme ist die grafische Darstellung (Zeigerinstrumente, Balken ...) von Werten besser geeignet.
Um Schwierigkeiten beim Farberkennungsvermögen des Bedieners vorzubeugen (Farbenblindheit), sollten wichtige Veränderungen eines Objekts (Zustand) sowohl durch eine Farb- wie auch durch eine Formänderung angezeigt werden.
Wichtige Informationen sollten in einem Bild sofort sichtbar sein. Dies setzt einen gezielten Einsatz von Farbkontrasten voraus.

Farbcodierung

Das menschliche Auge nimmt Farben schneller wahr als z.B. Text. Durch das Arbeiten mit einer sogenannten Farbcodierung bekommt man schneller einen Eindruck davon, in welchem Status (Zustand) sich die verschiedenen Objekte befinden. Es ist wichtig, daß man bei der Farbcodierung, was die Bedeutung der Farben angeht, konsequent vorgeht. Einheitliche Farbfestlegungen in einem Projekt für die Darstellung von Zuständen (z.B. rot für Störung) sind inzwischen Standard. Beim Kunden bereits vorhandene Firmenstandards müssen berücksichtigt werden.

Textdarstellung

Um die Lesbarkeit des Textes zu erhöhen, sollte man sich an einige einfache Regeln halten.

- Die Größe des Textes muß an die Wichtigkeit der Textinformation, aber auch an den zu erwartenden Abstand zwischen dem Bediener und dem Bildschirm angepaßt werden
- Verwenden Sie bevorzugt Kleinbuchstaben. Sie brauchen weniger Platz und sind besser zu lesen als Großbuchstaben, obwohl diese aus der Entfernung besser zu erkennen sind.
- Horizontaler Text ist besser lesbar als vertikaler oder schräggestellter Text.
- Benutzen Sie unterschiedliche Schriftarten für unterschiedliche Informationen (z.B. für Meßstellennamen, Hinweise, u.a.m.).

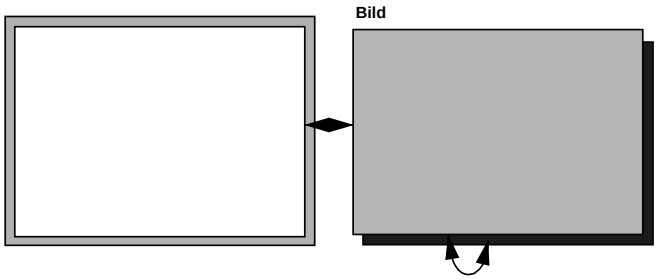
Das Konzept durchziehen

Egal für welches Konzept Sie sich entscheiden, Sie sollten es im gesamten Projekt einhalten. Sie unterstützen damit die intuitive Bedienung der Prozeßbilder. Fehlbedienungen werden unwahrscheinlicher.
Dies gilt auch für die verwendeten Objekte. Ein Motor oder eine Pumpe sollte immer gleich aussehen, egal in welchem Bild diese Objekte verwendet werden.

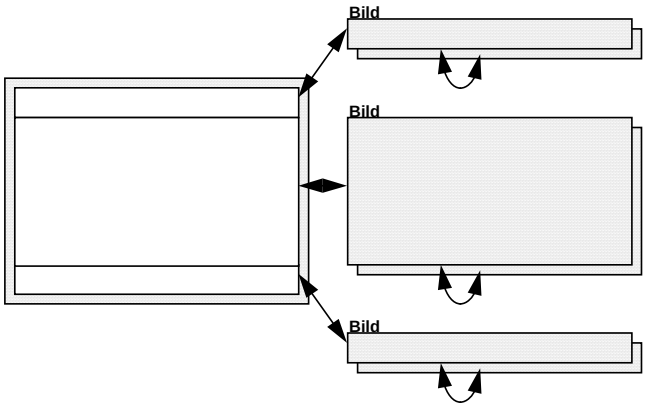
Einteilung des Bildschirms

Werden im Prozeß Standard PC-Monitore eingesetzt, so hat sich eine Aufteilung des Bildschirms in drei Bereiche (Übersichts-, Arbeits- und Tastenbereich) als sinnvoll erwiesen.
Läuft Ihre Anwendung jedoch auf einem speziellen Industrie-PC oder Operator Panel mit integrierten Funktionstasten, ist diese Unterteilung nicht immer sinnvoll.

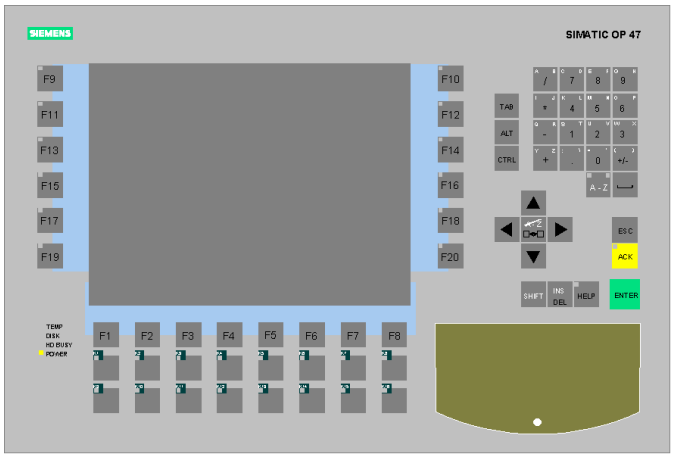
Bilder belegen den gesamten Bildschirmbereich



Der Bildschirm ist in Übersichts-, Tastenbereich und Anlagenbilder aufgeteilt



Beispiel eines Operator Panel



Parameter / Grenzen

Die Größe der einzelnen Bilder kann innerhalb fester Grenzen (min 1 x 1, max. 4096 x 4096 Pixel) frei gewählt werden. Bei Einplatzsystemen mit einem 17" Monitor ist eine maximale Auflösung von 1024 x 768 Pixel zu empfehlen. Bei Mehrplatzsystemen (Multi-VGA) kann eine größere Auflösung sinnvoll sein.
Bei Operator Panels beschränkt meist die vorhandene Technik die Auflösung (TFT von 640 x 480 bis 1024 x 768).

Festlegung

Für Bilder in den Projekten zu diesem Handbuch gilt:

Auflösung

Wir verwenden in unseren Projekten Auflösungen von 1024 x 768 und in Ausnahmefällen 800 x 600 Pixel. Für die korrekte Darstellung unserer Beispielprojekte setzen wir eine Einstellung der Farbpalette Ihres PC´s von mindestens 65536 Farben voraus.

Texte

Wir verwenden für Meßstellenbezeichnungen die Schriftart Courier, für reine Beschreibungen, für alle anderen Texte und Textdarstellungen die Schriftart Arial. Bei Hinweisboxen im Windows-Stil werden die Schriftarten MS Sans Serif und System verwendet.
Die Schriftgröße passen wir je nach Bedarf an.

Informationen im Bild

Immer wenn es sinnvoll erscheint, blenden wir Informationen aus Bildern aus. Wir zeigen diese Informationen nur bei Bedarf (Bedienung oder automatisch) an.
Wir verwenden in unseren Projekten mehrere unterschiedliche Bildschirmeinteilungen. Bei vielen bedienbaren Objekten geben wir über Tooltip Hinweise für die Bedienung.

Einteilung des Bildschirms

Wir werden die prinzipiellen Möglichkeiten der Bildschirmeinteilung projektieren. In den übrigen Projekten jedoch die Aufteilung in Kopfzeile, Arbeitsbereich und Tastenbereich anwenden.

Hinweis für die allgemeine Verwendung

Die Grundaufteilung der verwendeten Konzepte können Sie direkt für Ihre Projekte wiederverwenden.

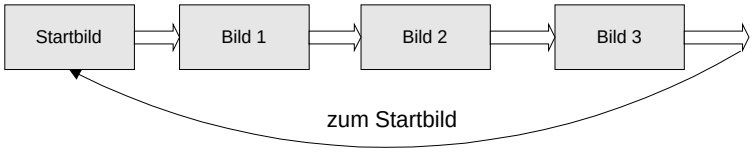
3.2.6 Festlegung: Das Bedienkonzept

Allgemeines

Für die Bedienung Ihrer Anwendung im Prozeß stehen in WinCC die bekannten Eingabemittel wie Tastatur, Maus, Touch Screen oder Industrie Joystik zur Verfügung. Für rauhe Industrieumgebungen, in denen der Einsatz einer Maus nicht möglich ist, können Sie die Bedienung über **Schalt-Cursor** und der **Alpha-Cursor** projektieren. Der Schalt-Cursor springt bedienbare Felder an, der Alpha-Cursor die Eingabefelder.
Jede Bedienung läßt sich gegen unberechtigte Zugriffe verriegeln.

Bildanwahl

Das Konzept für die Anwahl der Bilder hängt von mehreren Faktoren ab. Ausschlaggebend dafür ist die Anzahl der Bilder und die Struktur des Prozesses, der dargestellt werden soll. Für kleinere Anwendungen ist die Anordnung der Bilder als Umlaufpuffer möglich.



Bei einer größeren Anzahl von Bildern ist eine hierarchische Anordnung bei der Bildanwahl unerläßlich. Damit sich die Anwahl der Bilder durch das Bedienpersonal möglichst schnell erlernen läßt, ist eine einfache und feste Struktur zu wählen.

Die direkte Anwahl von Bildern ist natürlich auch möglich und bei Kleinstanwendungen (z.B. Kühlhaus) eventuell sinnvoll.

Hierarchie

Eine hierarchische Struktur macht den Prozeß überschaubar, einfach zu handhaben und gibt, falls notwendig, schnell Zugang zu detaillierten Informationen.
Eine gängige und häufig eingesetzte Hierarchie besteht aus drei Ebenen.

Ebene 1

In der Ebene 1 werden die Übersichtsbilder eingeordnet.
Diese Ebene enthält hauptsächlich Informationen darüber, welche unterschiedlichen Systemabschnitte es im System gibt und wie diese Systemabschnitte zusammenarbeiten.
Hier wird ebenfalls gezeigt, ob ein Ereignis (Meldung) in tieferliegenden Ebenen aufgetreten ist.

Ebene 2

In der Ebene 2 werden die Prozeßbilder eingeordnet.
Diese Ebene enthält detaillierte Informationen über einen bestimmten Prozeßabschnitt und zeigt, welche Anlagenobjekte zu diesem Prozeßabschnitt gehören.
Hier wird ebenfalls gezeigt, auf welches Anlagenobjekt sich ein Alarm bezieht.

Ebene 3

In der Ebene 3 werden die Detailbilder eingeordnet.
Diese Ebene gibt Informationen über einzelne Anlagenobjekte, z.B. Regler, Ventile, Motoren etc. Hier werden Meldungen, Zustände und Prozeßwerte gezeigt. Eventuell werden auch Informationen über das Zusammenwirken mit anderen Anlagenobjekten gegeben.

Festlegung

Für Projekte, die im Zuge der Erstellung dieses Handbuchs entstehen, gilt folgende Festlegung:
Wir werden in unseren Projekten mehrere unterschiedliche Bedienkonzepte verwenden und deren Unterschiede aufzeigen.

Hinweis für die allgemeine Verwendung

Für den Aufbau eines eigenen Bedienkonzeptes können unsere Projekte nur als Anregung dienen. Bei Erweiterungen von Anlagen muß auf bestehende Bedienkonzepte Rücksicht genommen werden. Bei vielen Anwendern gibt es inzwischen Firmenstandards, die bei der Projektierung einzuhalten sind.

Hinweis:
Im WinCC Optionspaket **Basic Process Control** wird ein vorgefertigtes Bedienkonzept angeboten. Das Optionspaket beinhaltet darüber hinaus noch weitere nützliche und leistungsfähige Funktionen (z.B. Storage).

3.2.7 Festlegung: Die Farbdefinition

Allgemeines

Über Farben wird im Umfeld von HMI-Systemen viel und gerne diskutiert. In WinCC können Sie die Farben für Linien, Rahmen, Hintergrund, Füllmuster, und Schrift frei wählen. Es stehen Ihnen alle von Windows unterstützten Farben zur Verfügung. Selbstverständlich lassen sich in WinCC während der Laufzeit die Farben und natürlich auch die anderen Grafikeigenschaften verändern.
Für eine kostengünstige Projektierung und eine übersichtliche Darstellung der Prozesse kommt der Farbdefinition eine besondere Bedeutung zu.

Für folgende Bereiche sollte immer eine Definition der Farben durchgeführt werden. Die Festlegung kann eventuell nach DIN *EN 60073* entspricht *VDE 0199* erfolgen, muß aber immer mit dem Anwender abgestimmt werden:

- Farben für Meldungen (gekommen / gegangen / quittiert)
- Farben für Zustände (ein / aus / gestört)
- Farben für Zeichenobjekte (Leitungen / Füllstände)
- Farben für Warn- und Grenzwerte

Festlegung

Für Farben in den Projekten zu diesem Handbuch gilt:
Für eine korrekte Darstellung der Beispielprojekte ist eine Einstellung größer 256 Farben zu wählen.
In den Beispielprojekten verwenden wir, zur besseren Orientierung, für die einzelnen Themen (Variablen, C-Kurs, Bildprojektierung) jeweils eine eigene Hintergrundfarbe. Im Übersichts- und Tastenbereich ist die Hintergrundfarbe dunkler.
Im Meldesystem ist jeder Meldeklasse und jeder einer Meldeklasse zugeordneten Meldeart ein bestimmter Farbcode zugeordnet.

Hinweis für die allgemeine Verwendung

Nach der Festlegung der Farben sollten Sie gegebenenfalls die Defaulteinstellungen von WinCC anpassen.

Eine Tabelle zur Codierung der Farbwerte in *C-Aktionen* finden Sie im Anhang Kapitel 5.1.6 Farbtabelle.

3.2.8 Festlegung: Die Aktualisierungszyklen

Allgemeines

Bei der Festlegung der Aktualisierung ist immer die Betrachtung des Gesamtsystems entscheidend. Was wird aktualisiert und wie oft wird aktualisiert. Die falsche Wahl der Aktualisierungszyklen können negative Einflüsse auf die Performance des HMI-Systems haben.
Bei der Betrachtung eines Gesamtsystems (PLC - Kommunikation - HMI), sollten Änderungen dort erkannt werden, wo sie entstehen, nämlich im Prozeß (PLC). In vielen Fällen stellt das Bussystem den Engpaß für die Datenübertragung dar.

Bei der Festlegung der Aktualisierung von Meßwerten ist darauf zu achten, wie schnell sich der Meßwert tatsächlich ändert. Bei der Temperaturregelung für einen Kessel mit ca. 5.000l Inhalt macht eine Aktualisierung des Istwertes im 500ms Takt absolut keinen Sinn.

32Bit HMI-System

WinCC ist ein reines 32Bit HMI-System, basierend auf Windows NT. Diese Betriebssysteme sind optimiert für ereignisgesteuerte Bedienung. Wenn Sie diesen Grundsatz bei der Projektierung mit WinCC berücksichtigen, werden Performanceprobleme, auch bei sehr großen Datenmengen, eher die Ausnahme sein.

Festlegung

Für die Aktualisierung in den Projekten zu diesem Handbuch gilt:
Die Aktualisierung erfolgt, soweit es die Aufgabenstellung erlaubt, möglichst ereignisgesteuert. Da wir überwiegend mit *internen Variablen* arbeiten triggern wir oft auf Änderung der Variablen. Bei der Verwendung von externen Variablen kann dies, abhängig vom Prozeßanschluß, zu erhöhter Systembelastung führen. Erlaub die Kommunikation eine ereignisgesteuerte Übertragung, so ist dies für zeitkritische Daten zu wählen. Unkritische Daten können dann vom HMI in angemessenen Zyklen geholt werden (Pollingverfahren).

Hinweis für die allgemeine Verwendung

Eine ausführliche Beschreibung für die Verwendung der Aktualisierungszyklen finden Sie im Kapitel 3.3.1 Aktualisierungszyklen - Wie und wo erfolgt die Einstellung.

3.2.9 Festlegung: Die Benutzerrechte

Allgemeines

Im Anlagenbetrieb ist es erforderlich, daß bestimmte Bedienfunktionen vor unberechtigtem Zugriff geschützt werden. Eine weitere Forderung ist, daß nur bestimmte Personen auf das Konfigurationssystem zugreifen dürfen.
Im *User Administrator* können Benutzer und Benutzergruppen angelegt und verschiedene Berechtigungsstufen definiert werden. Diese Berechtigungsstufen können mit den Bedienelementen in den Bildern verbunden werden.
Den Benutzergruppen und Benutzern können individuell verschiedene Berechtigungsstufen zugewiesen werden.

Festlegung

In den Beispielprojekten steht jedem Benutzer die Projektbedienung frei.

Im Beispielprojekt *Project_CreatePicture* ist die Bedienung im Projekt jedoch erst nach Anmeldung möglich. Das erforderlicht Paßwort lautet *pictu_00*. Die Buttons für die Anwahl der einzelnen Themen sind dort mit der Berechtigungsstufe *Projektbedienung* verbunden.

Hinweis für allgemeine Verwendung

Eine Beschreibung für die Vergabe der Benutzerrechte finden Sie im Configuration Manual Band 2 im Beispielprojekt *Project_CreatePicture* im Kapitel Bedienfreigaben.

3.2.10 Festlegung: Die Alarmierung

Allgemeines

WinCC unterstützt grundsätzlich zwei Meldeverfahren:

- Das **Bitmeldeverfahren** ist ein universelles Verfahren, das Meldungen aus beliebigen Automatisierungssystemen erlaubt. WinCC überwacht den Flankenwechsel ausgewählter Binärvariablen selbst und leitet daraus Meldungsereignisse ab.
- Das **Zeitfolgerichtige Melden** setzt voraus, daß die Automatisierungssysteme die Meldungen selbst bilden und mit Zeitstempel und eventuell mit Prozeßwerten in einen vordefinierten Format an WinCC senden. Die chronologische Ordnung von Meldungen unterschiedlicher Automatisierungssysteme wird erst durch dieses Meldeverfahren möglich. Siehe Kapitel 5.2 Dokumentation des S5-Meldesystems.

Was soll gemeldet werden?

Bei der Festlegung, welche Ereignisse und Zustände gemeldet werden, geht man oft den vermeintlich sicheren Weg und meldet alle Ereignisse und Zustandsänderungen. Damit wird dem Bediener die Entscheidung überlassen welche Meldungen er sich zuerst ansieht. Werden in einer Anlage zu viele Ereignisse gemeldet, passiert es erfahrungsgemäß, daß wichtige Meldungen erst zu spät beachtet werden.

Hinweis für allgemeine Verwendung

Die Darstellung der Meldungen und die Auswahl der Meldungen für die Archivierung kann geändert und den eigenen Anforderungen angepaßt werden.

3.2.11 Festlegung: Zur Realisierung

Allgemeines

Für die Realisierung eines Projekts ist es sinnvoll, für die Ablage der Daten eine feste Struktur zu verwenden. Die Festlegung beginnt mit der Entscheidung, auf welchem Laufwerk das WinCC Projekt angelegt wird. Der nächste Schritt betrifft den Aufbau der Verzeichnisstruktur, u.a.m.
Nach unseren Erfahrungen ist es sinnvoll, alle Daten eines Projekts unter einem Verzeichnis mit entsprechenden Unterverzeichnissen abzulegen. Dieses Vorgehen hat Vorteile bei der Projektbearbeitung, aber vor allem bei der Datensicherung.

Hinweis:
Die Ausstattung der PCs ist sehr unterschiedlich. Um sich bei der Vergabe des Ziellaufwerks für die Projektbearbeitung von diesen Abhängigkeiten zu lösen, bieten sich **virtuelle Laufwerke** an. Die Zuordnung des Verzeichnisses zum virtuellen Laufwerk kann jederzeit geändert werden.

Festlegung Verzeichnisse

Neben den Verzeichnissen, die durch WinCC erzeugt werden, legen Sie bei Bedarf weitere Verzeichnisse für Word, Excel und für temporäre Dateien an.

3.3 Besonderheiten bei der Projektierung mit WinCC

In den nachfolgenden Kapiteln werden übergreifende Themen für die Projektierung mit WinCC behandelt.
Diese Themen sind eine Ergänzung zur Hilfe von WinCC.

3.3.1 Aktualisierungszyklen - Wie und wo erfolgt die Einstellung

Die Festlegung der Aktualisierungszyklen ist eine der bedeutendsten Einstellungen im Visualisierungssystem. Aufgrund der Festlegungen werden folgende Eigenschaften beeinflußt:

- der Bildaufbau
- die Aktualisierung der Objekte des aktuell aufgeschlagenen Bildes an der Visualisierungsstation (*Graphics Designer*)
- die Bearbeitung von Hintergrundscripten (*Global Script*)
- die Aktivierung des Datenmanagers sowie die Prozeßkommunikation

Weitere Zeitgrößen werden in der Meßwertverarbeitung (*Tag Logging*) unter den Archivierungszeiten eingestellt.

Datenmanager

Abhängig von den eingestellten Aktualisierungszyklen werden die aktuellen Variablenwerte jeweils vom Datenmanager, dem zentralen Verwalter des Variablenhaushalts, angefordert. Siehe Kapitel 3.3.2 WinCC Dynamisierung.

Der Datenmanager ermittelt die neuen Prozeßdaten über die Kommunikationskanäle und versorgt die Applikationen mit diesen aktuellen Werten. Das Anfordern von Daten bedeutet somit jeweils einen Wechsel zwischen verschiedenen Tasks (*Graphics Designer*, *Datenmanager* etc.). Abhängig von der Projektierung kann dies zu sehr unterschiedlichen Systembelastungen führen.

3.3.1.1 Die Aktualisierung im Bild

Bildaktualisierung

Die Aktualisierung der einzelnen Eigenschaften der Objekte im Bild bezieht sich nach dem Bildaufschlag auf die dynamisierten Objekte. Über den Aktualisierungszyklus wird die Aktualität der Eigenschaften des jeweiligen Objektes im Bild bestimmt. Der Aktualisierungszyklus der dynamisierten Objekte kann bei folgenden Dynamisierungsarten vom Projekteur bzw. vom System eingestellt werden:

Dynamisierungsart	**Standard-Einstellung**	**Projektierungsanpassung**
Konfigurationsdialog	Variablentrigger 2 Sek. Oder Ereignistrigger (z.B. Bedienung)	Anpassung der Zeitzyklen
Dynamic-Wizard	Abhängig von der Art der Dynamisierung besteht die Auswahl aus • Ereignistrigger, • Zeitzyklus • Variablentrigger	Anpassung der Zeitzyklen, Ereignisse oder Variablen
Direktverbindung	Ereignistrigger	
Variablenanbindung	Variablentrigger 2 Sek.	Anpassung der Zeitzyklen
Dynamik-Dialog	Variablentrigger 2 Sek.	Anpassung der Zeitzyklen, Variablentrigger
C-Aktion für Eigenschaften	Zeitzyklus 2 Sek.	Anpassung der Zeitzyklen, Variablentrigger direktes Lesen von der Steuerung
Objekt-Eigenschaft	Einstellung abhängig von der Dynamisierung	Bearbeitung der Spalte Aktualisierungszyklus

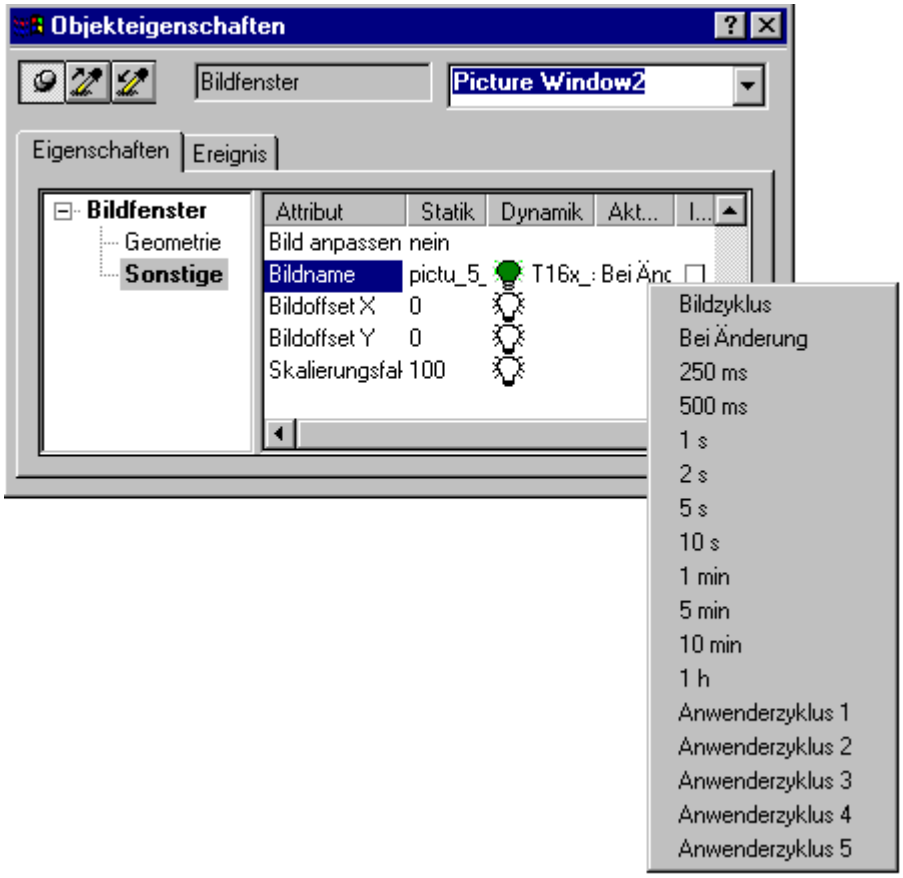
Die auszuwählenden Aktualisierungszyklen sind von WinCC vorgegeben und können durch eigene anwenderdefinierte Zeitzyklen ergänzt werden.

3.3.1.2 Arten der Aktualisierungszyklen

Bei den Aktualisierungszyklen werden folgende Arten unterschieden:

Art	Standard-Einstellung
Standardzyklus	Zeitzyklus von 2 Sekunden
Zeitzyklus	2 Sekunden
Variablentrigger	2 Sekunden
Bildzyklus	2 Sekunden
Fensterzyklus	bei Änderung
benutzerdefinierte Zeitzyklen	Anwenderzyklus1: 2 Sek Anwenderzyklus2: 3 Sek Anwenderzyklus3: 4 Sek Anwenderzyklus4: 5 Sek Anwenderzyklus5: 10 Sek

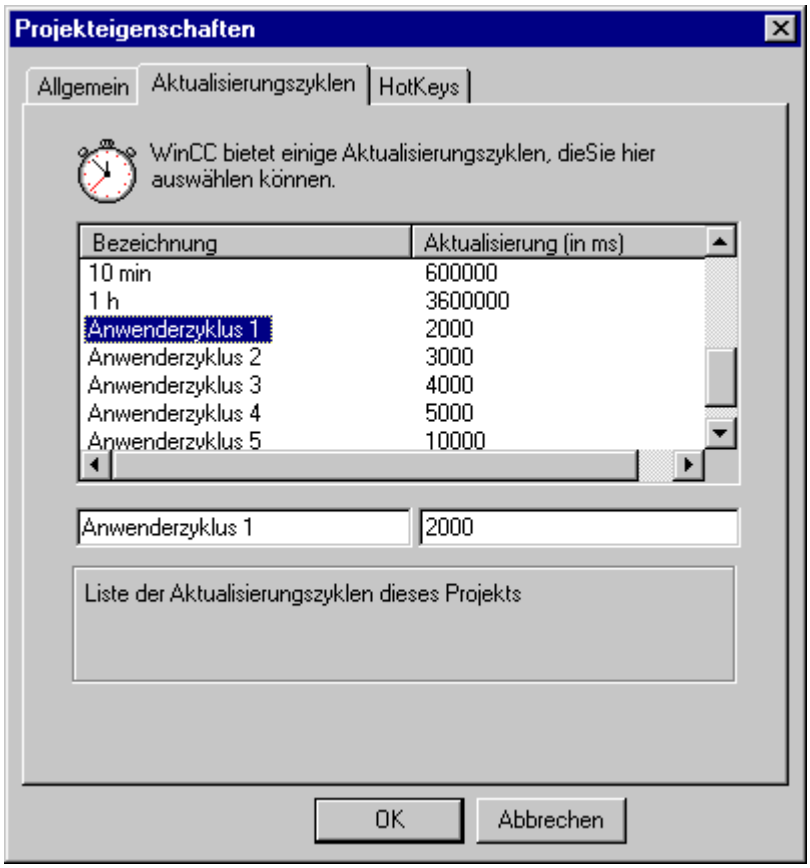
Auswahl der Aktualisierungszyklen, z.B. für die Eigenschaft eines Objektes:



Anwenderzyklus

Bis zu 5 Anwenderzyklen können **projektbezogen** definiert werden. Ist im *WinCC Explorer* in der linken Baumstruktur der Projektname selektiert, kann über den nachfolgend dargestellten Button der Symbolleiste der Dialog *Projekteigenschaften* geöffnet werden.

Im Dialog *Projekteigenschaften* auf der Registerkarte *Aktualisierungszyklen* werden am Ende der Liste der eingestellten Standard-Aktualisierungszyklen die Anwenderzyklen 1...5 für die projektbezogene Definition angeboten. Nur diese Anwenderzyklen können parametriert werden.



Über diesen Weg können Zeitzyklen, die noch nicht als Zeiteinheiten zur Verfügung stehen (z.B. 200 ms) definiert werden.
Der Anwenderzyklus kann von 100 ms bis zu 10 Stunden festgelegt werden. Die Bezeichnung für die Anwenderzyklen ist frei wählbar.

Diese projektbezogenen Zeiteinheiten können für ausgewählte Objekte eingesetzt werden, deren Aktualisierungszyklus zu einem späteren Zeitpunkt geändert werden muß. Eine Optimierungsmaßnahme könnte ein Grund für die Änderung der Zeitzyklen sein. Die anwenderdefinierten Aktualisierungszyklen eignen sich dazu, nachträglich an nur **einer** zentralen Stelle den eingestellten Zeitzyklus zu modifizieren. Die einzelnen Objekte der Bilder müssen dann nicht mehr zusätzlich angepaßt werden. Dieser Weg der Definition von Anwenderzyklen sollte daher für **wartungsfreundliche Projekte** in Betracht gezogen werden.

3.3.1.3 Bedeutung der Aktualisierungszyklen

Vor dem Einsatz der möglichen Aktualisierungszyklen muß zunächst die Bedeutung der verschiedenen Aktualisierungszyklen betrachtet werden.

Bei den Aktualisierungszyklen werden folgende Arten unterschieden:

Art	**Bedeutung**
Standardzyklus	Zeitzyklus
Zeitzyklus	Nach der jeweils eingestellten Zeit wird die Eigenschaft bzw. Aktion des einzelnen Objektes aktualisiert. Dies bedeutet, daß die Variablen jeweils **einzeln** vom Datenmanager angefordert werden.
Variablentrigger	Abhängig von der eingestellten Zykluszeit werden jeweils nach Ablauf der Zeitspanne die Variablen vom System ermittelt und gegen Wertänderung geprüft. Ändert sich in dem eingestellten Zeitrahmen der Wert mindestens einer ausgewählten Variable, so dient dies als Trigger für die davon abhängigen Eigenschaften bzw. Aktionen. **Alle** Variablenwerte werden **gemeinsam** vom Datenmanager angefordert.
Bildzyklus	Aktualisierung der Eigenschaften des aktuellen Bildobjektes sowie aller Objekte, die über den Aktualisierungszyklus Bildzyklus getriggert werden.
Fensterzyklus	Aktualisierung der Eigenschaften des Fensterobjektes sowie aller Objekte die über den Aktualisierungszyklus Fensterzyklus getriggert werden.
benutzerdefinierte Zeitzyklen	Zeiteinheiten, die projektbezogen definiert werden können.
C-Aktion für direktes Lesen von der Steuerung	Mittels interner Funktionen in den *C-Aktionen* können Werte direkt von der Steuerung gelesen werden. Die weitere Bearbeitung der Folgebefehle in der *C-Aktion* wird erst nach dem Lesen der Prozeßwerte fortgesetzt (synchrones Lesen).

Hinweis:
Das Anfordern des aktuellen Variablenwertes vom Datenmanager führt jeweils zum Taskwechsel und einem Datenaustausch zwischen den einzelnen Tasks. Zusätzlich müssen die Variablenwerte vom Datenmanager über den Kommunikationskanal der angebundenen Automatisierungsgeräte angefordert werden. Dies erfolgt abhängig von der Art der Kommunikation über Anforderungstelegramme an die Kommunikationsschnittstelle (FETCH) und Datentelegramme von der Automatisierungseinheit zurück an WinCC.

3.3.1.4 Hinweise zur Verwendung der Aktualisierungszyklen

Bei Verwendung der Aktualisierungszyklen wird abhängig von der jeweiligen Art folgende Einstellung empfohlen:

Art	Standard-Einstellung	Empfehlung für die Projektierung
Standardzyklus	Zeitzyklus von 2 Sekunden	*Dynamik-Dialog* oder *C-Aktion*: falls eine Abhängigkeit von Variablen vorliegt sollte auf jeden Fall eine Variablentriggerung eingesetzt werden. Dies führt zu einer Reduzierung der Taskwechsel sowie der Kommunikation zwischen den Tasks. Die Variablentriggerung *bei Änderung* darf nur gezielt eingesetzt werden, da dies zu einer höheren Systembelastung führen kann! Die Variablen werden in diesem Fall ständig auf Änderung geprüft. Dieser Polling-Mechanismus führt immer zu einer höheren Systembelastung. Bei Standard-Objekten wird ein Zyklus von 1 bis 2 Sekunden empfohlen.
Zeitzyklus	2 Sekunden	Den Zeitzyklus jeweils vom Objekttyp bzw. der Objekteigenschaft abhängig machen. Die Trägheit von Prozeßkomponenten (Tankfüllungen oder Temperaturen im Gegensatz zu Schalthandlungen) sollte ebenfalls berücksichtigt werden. Bei Standard-Objekten wird ein Zyklus von 1 bis 2 Sekunden empfohlen.
Variablen-trigger	2 Sekunden (bei *Dynamik-Dialog*)	Falls diese Aktualisierungsmöglichkeit projektierbar (abhängig von der Dynamisierungsart) ist, sollte diese **vorzugsweise** eingesetzt werden! Berücksichtigen Sie bei einer Abhängigkeit von Variablen auf jeden Fall **alle** Variablen, die für eine Änderung der Eigenschaft bzw. für den Aktionsdurchlauf verantwortlich sind. Nur die in der Liste aufgeführten Variablen dienen als Trigger für die Aktualisierung der dynamisierten Eigenschaft bzw. Aktion. Die **Variablentriggerung** *bei Änderung* sollte nur gezielt eingesetzt werden. Sobald sich eine der ausgewählten Variablen geändert hat, wird der Trigger für diese Eigenschaft bzw. Aktion ausgelöst. Dieser Polling-Mechanismus führt zu einer höheren Systembelastung.
Bildzyklus	2 Sekunden	Dieser sollte nur dann verkleinert werden, wenn sich die dynamisierten Eigenschaften des Bildobjektes selbst in einem kürzeren Zeitraum ändern und damit aktualisiert werden müssen. Eine Erhöhung dieses Bildzyklus verringert die Systembelastung.

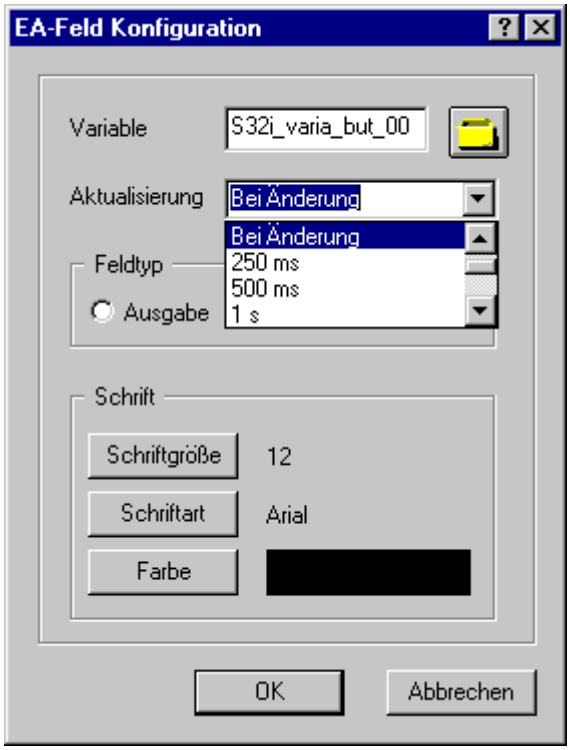
Art	Standard-Einstellung	Empfehlung für die Projektierung
Fensterzyklus	bei Änderung	Handelt es sich um ein Bildfenster, das z.B. zum Verstellen von Prozeßgrößen eingeblendet wird (Prozeßbox), so ist diese Einstellung sinnvoll. Wird das Bildfenster für Informationszwecke ständig angezeigt (z.B. Bildschirmgliederung), so sollte die Aktualisierung des Fensters sowie der Inhalte auf Variablentrigger oder Zeitzyklus eingestellt werden.
C-Aktion für direktes Lesen von der Steuerung		Die Internen Funktionen (z.B. GetTagWordWait) zum synchronem Lesen von Prozeßwerten (direkt von der Steuerung) sollten nur gezielt eingesetzt werden. Der Einsatz dieser Funktionen erfordert ein Pollen durch das System (Aktionssteuerung) und führt damit zu einer erhöhten Kommunikationsbelastung.

Einstellung der Aktualisierungszyklen

Die folgenden Beispiele zeigen, wo die Aktualisierungszyklen jeweils eingestellt werden:

Konfigurationsdialog

Dieser Dialog erscheint bei der Projektierung eines *Smart-Objektes* → *E/A-Feld*. Er kann jedoch auch über **R** über dem entsprechenden Objekt aufgerufen werden.

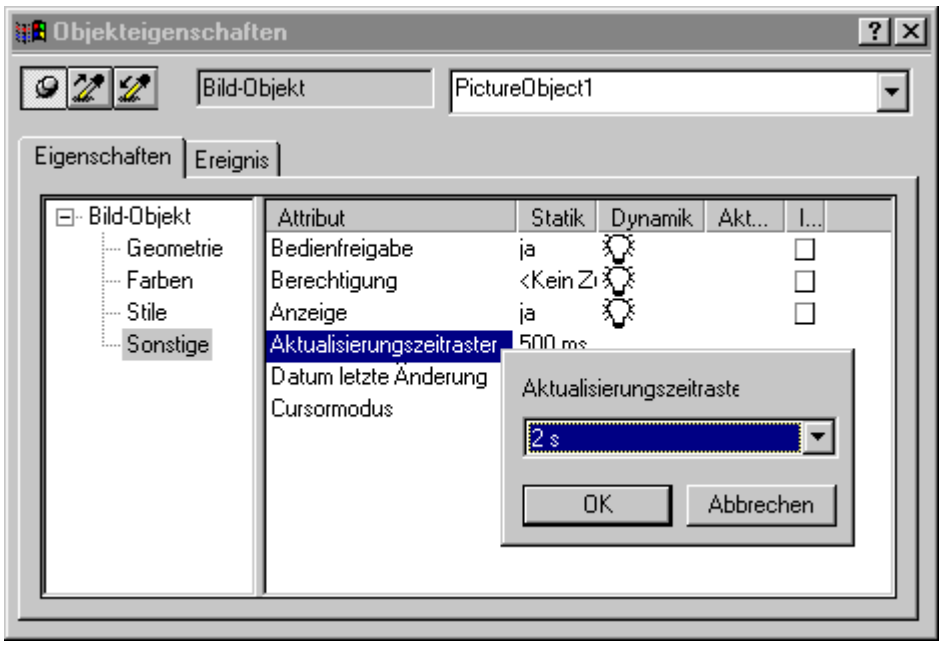


Dynamic Wizard

Diese Seite erscheint bei Anwahl des Punktes *Eigenschaft dynamisieren* auf der Registerkarte *Standard Dynamiken* im *Dynamic-Wizard* mit **D**.

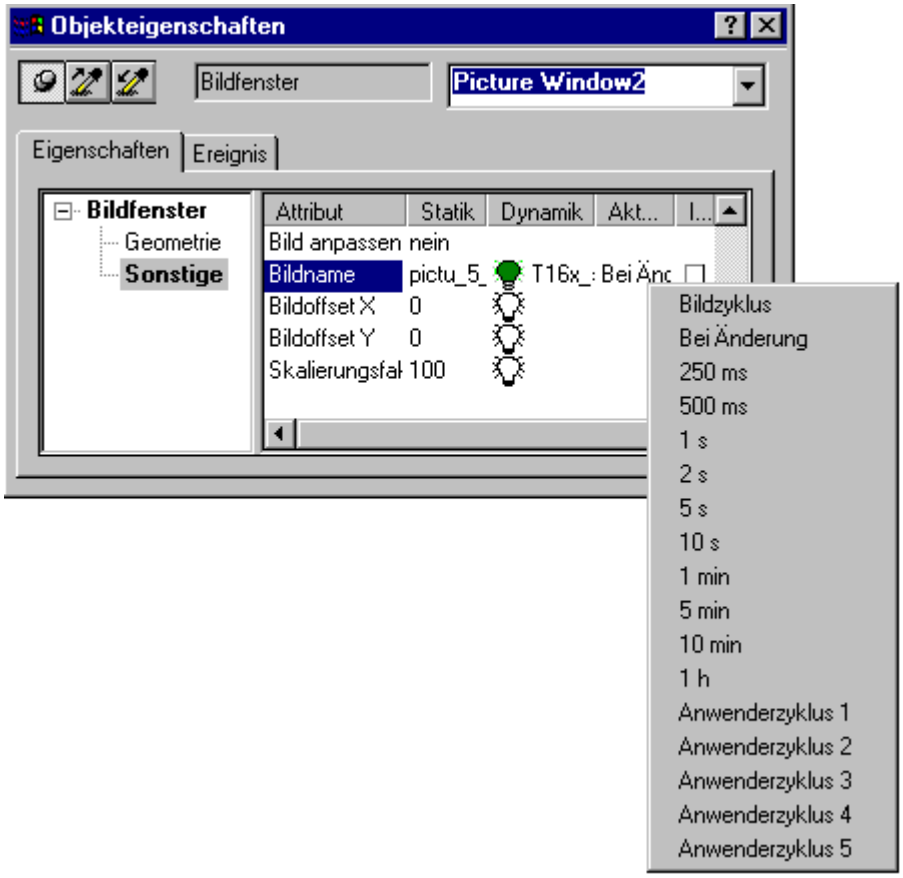
Bildzyklus

Ändern des Bildzyklus:



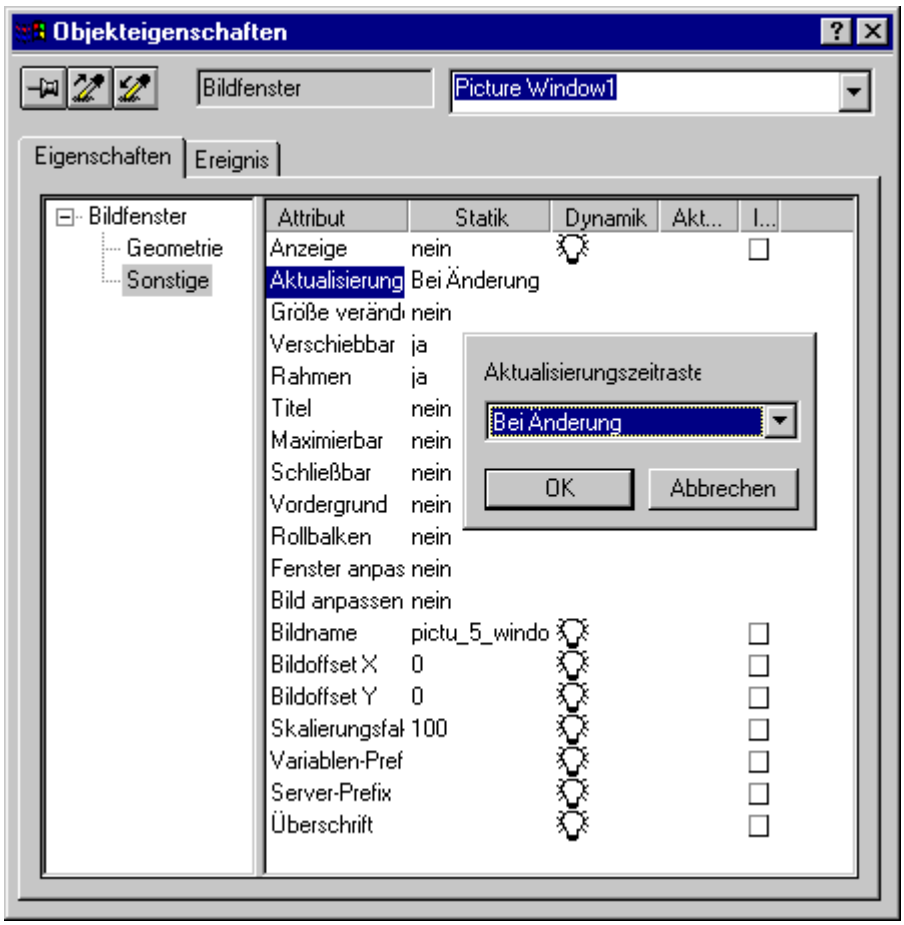
Variablenanbindung bei Objekt-Eigenschaft

Dieses Menü erscheint bei Anwahl der Spalte *Aktual.* mit **R** bei einer mit einer Variable dynamisierten Objekteigenschaft.



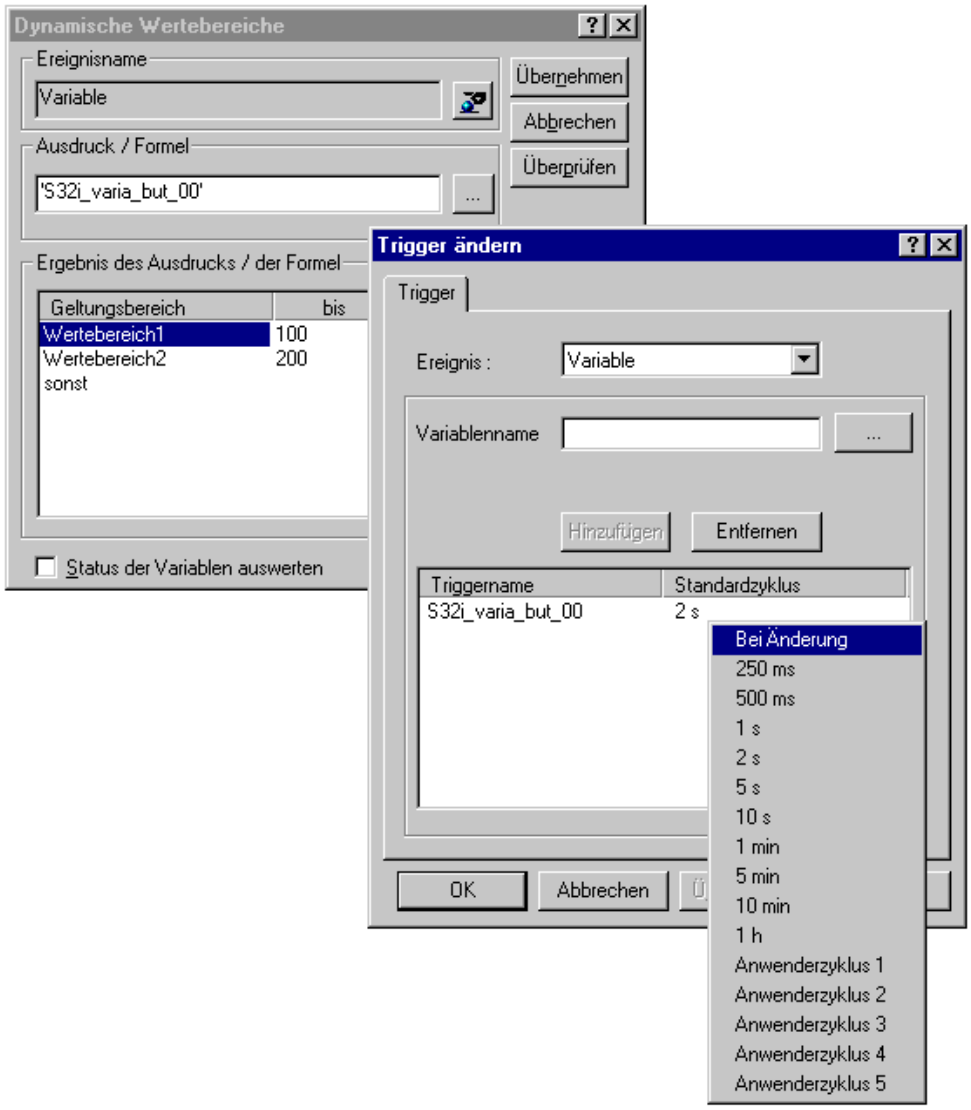
Fensterzyklus

Ändern des Fensterzyklus:



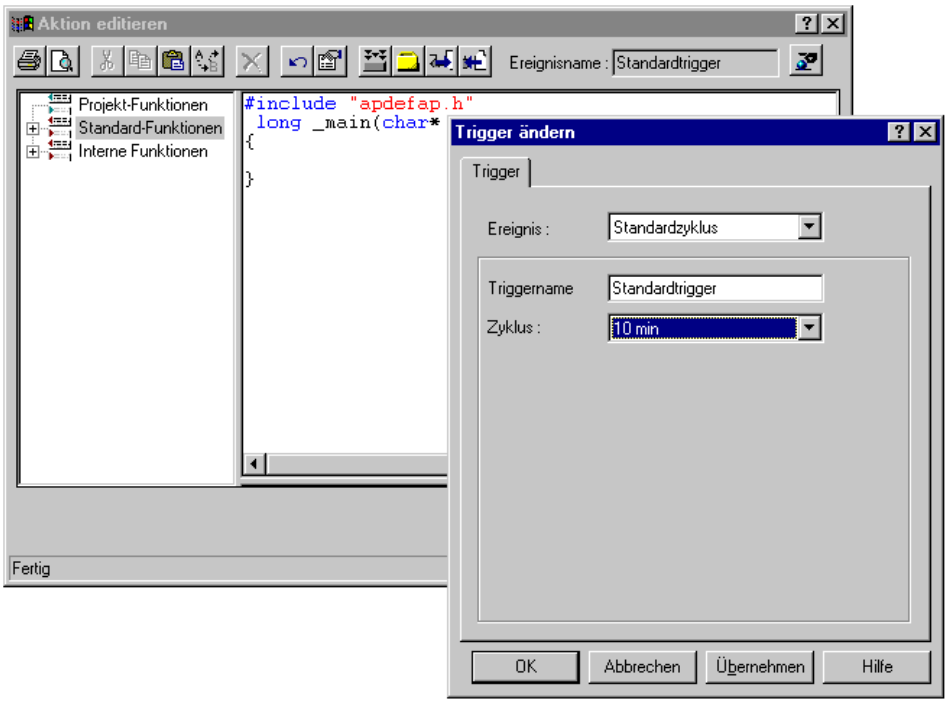
Dynamik-Dialog

Bei *Dynamik-Dialogen* wird durch Anwahl des Trigger-Button der Dialog zur Änderung vom Aktualisierungszyklus angewählt.



C-Aktion für Eigenschaft

Bei *C-Aktion* wird im Editor durch Anwahl des Trigger-Button der Dialog zur Änderung vom Aktualisierungszyklus angewählt.



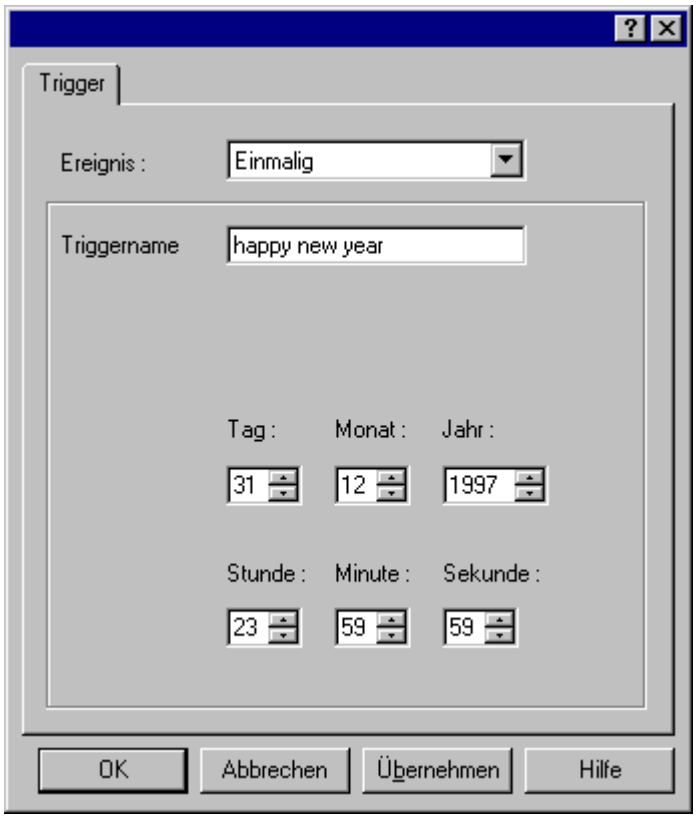
3.3.1.5 Ausführung von Hintergrundscripten (Global Script)

Die Ausführung von Hintergrundscripten (*Global Script*) ist je nach Projektierung von verschiedenen Größen abhängig:

- Timer (zyklische oder azyklische Ausführung)
  - Zeitzyklus
  - Zeitpunkt
- Variable (Ereignistrigger)

Timer - azyklisch

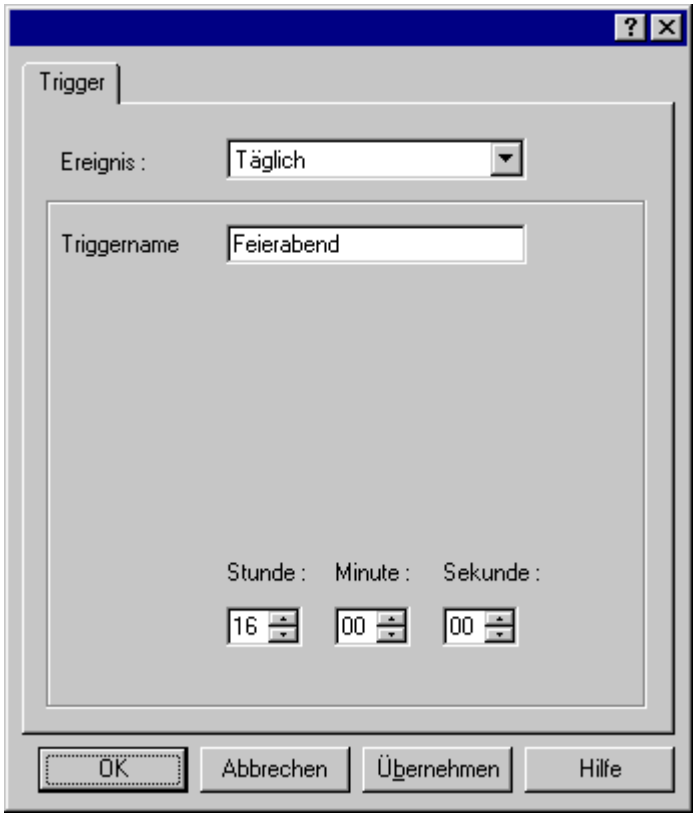
Einmalige Ausführung:



Timer - zyklisch

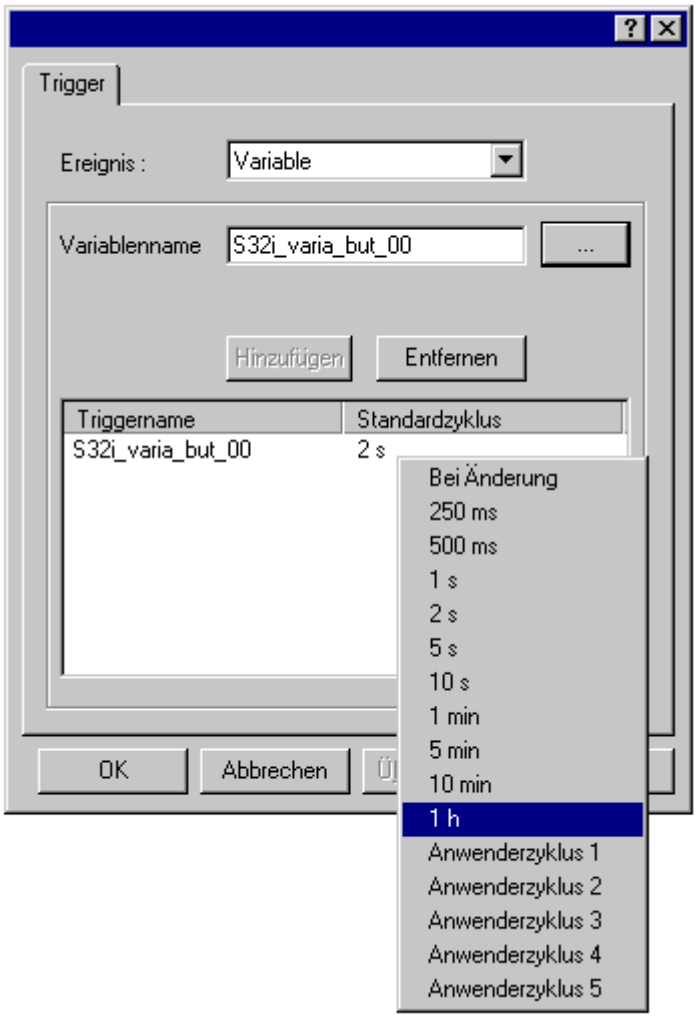
Der projektierte Zeitfaktor der globalen Aktion bestimmt, **wann** die definierte Aktionsfolge bearbeitet werden muß. Neben dem bereits beschriebenen Standardzyklus und den zugehörigen Zeiteinstellungen von 250 ms bis 1 h (bzw. Anwenderzyklus1 bis 5) , können auch die folgenden Zeit-Trigger gewählt werden:

- Stündlich (Minute und Sekunde)
- Täglich (Stunde, Minute, Sekunde)
- Wöchentlich (Wochentag, Stunde, Minute, Sekunde)
- Monatlich (Tag, Stunde, Minute, Sekunde)
- Jährlich (Monat, Tag, Stunde, Minute, Sekunde)



Variablentrigger

Wird die Aktion abhängig von einer oder mehreren Variablen aktiviert, so muß der Ereignistrigger als Variablentrigger gesetzt werden. Dies erfolgt analog dem Variablentrigger bei Objekteigenschaften. Standardmäßig wird als Zeitrahmen der Zyklus von 2 Sekunden gesetzt. Der Projekteur kann anstatt des Default-Wertes folgende Zeitfaktoren einstellen:



Zu Beginn und am Ende des eingestellten Zeitrahmens wird der Wert der ausgewählten Variablen ermittelt. Hat sich der Wert mindestens einer Variablen geändert, so wird der Trigger für die globale Aktion ausgelöst.
Beachten Sie die hohe Systembelastung bei Triggerung auf Variable *bei Änderung*. Diese Einstellung ist nicht immer sinnvoll. Es gelten die gleichen Hinweise wie bei der Objekt Aktualisierung.

Alle Aktionen, die Sie als globale Aktionen definieren, werden **nicht objektgebunden**, d.h. nur abhängig von den eingestellten Zeitzyklen oder Ereignistriggern, geprüft und aktiviert. Setzen Sie daher die globalen Aktionen gezielt ein und vermeiden Sie unnötige Aktionsschritte, um das System nicht zu stark zu belasten. Verwenden Sie nicht zu viele und zu viele kleine Zeitzyklen für die Ausführung Ihrer Aktionen.

3.3.2 WinCC Dynamisierung

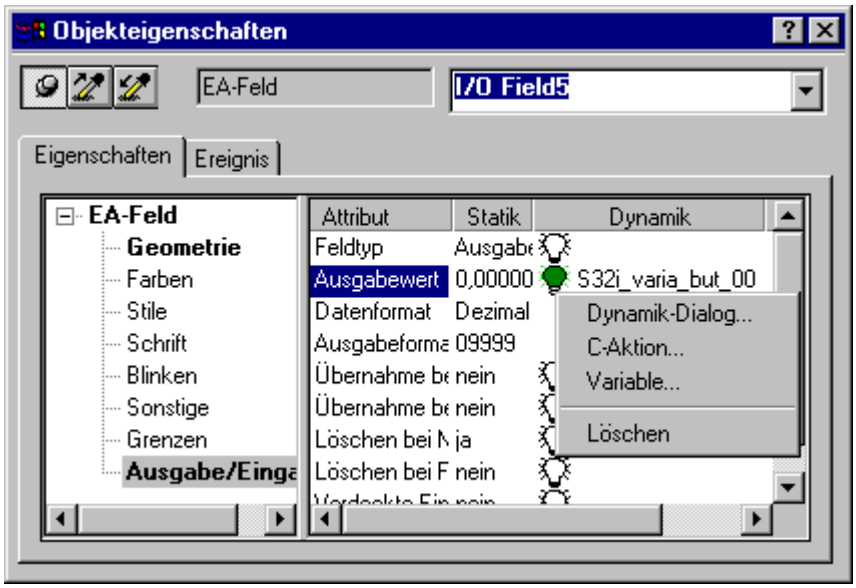
Definition

Unter Dynamisierung verstehen wir das Ändern von Zuständen (z.B. Position, Farbe, Schrift, u.a.m.) und das Reagieren auf Ereignisse (z.B. Mausklick, Tastaturbedienung, Wertänderung, u.a.m.) während der Laufzeit (Runtime).

Jedes Element im Grafikfenster wird als eigenständiges Objekt betrachtet. Das Grafikfenster selbst ist ebenfalls ein Objekt vom Typ *Bild-Objekt*.
Im WinCC *Grafik System* besitzt jedes Objekt *Eigenschaften* und *Ereignisse*. Mit wenigen Ausnahmen können diese dynamisiert werden. Diese Ausnahmen betreffen im wesentlichen *Eigenschaften* und *Ereignisse*, die im Runtime keine Auswirkung haben. Sie besitzen kein Symbol für die Dynamisierbarkeit.

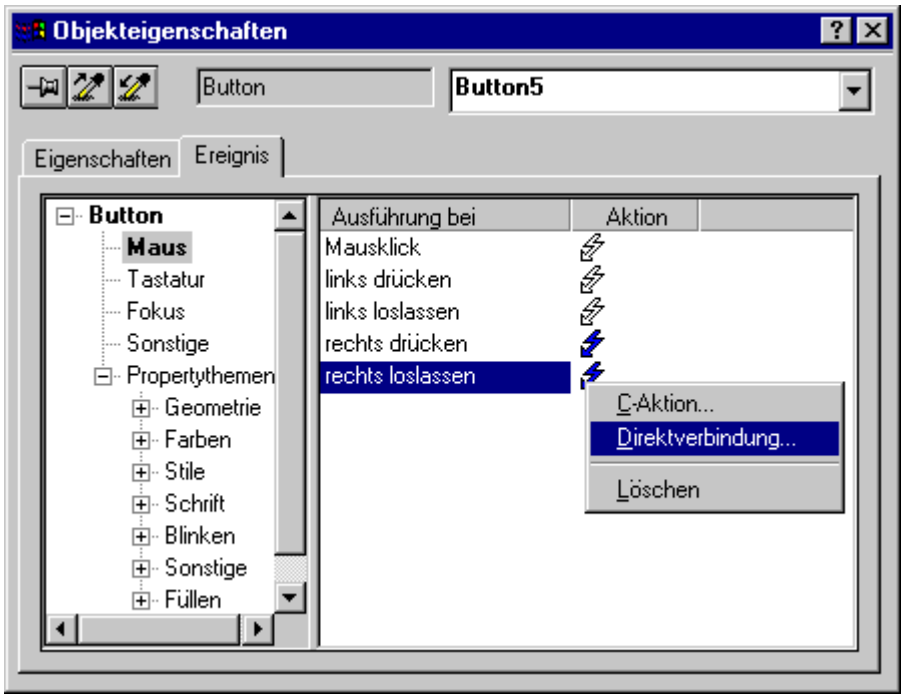
3.3.2.1 Dynamisieren der Eigenschaften

Die Eigenschaften eines Objekts (Position, Farbe, Schrift, u.a.m.) können statisch gesetzt und im Runtime dynamisch verändert werden.
Alle Eigenschaften mit einer Glühbirne in der Spalte *Dynamik* sind dynamisierbar. Ist eine Eigenschaft dynamisiert, wird abhängig von der Dynamisierungsart ein farbiges Symbol anstelle der weißen Glühbirne angezeigt. Themen (z.B. Geometrie), die dynamisiert sind, werden Fett dargestellt.



3.3.2.2 Dynamisieren der Ereignisse

Die Ereignisse eines Objekts (z.B. Mausklick, Tastaturbedienung, Wertänderung, u.a.m.) können im Runtime abgefragt und dynamisch ausgewertet werden.
Alle Ereignisse mit einem Pfeilsymbol in der Spalte *Aktion* sind dynamisierbar. Ist ein Ereignis dynamisiert wird abhängig von der Dynamisierungsart ein farbiger Pfeil anstelle des weißen Pfeils angezeigt. Themen (z.B. Sonstige), die dynamisiert sind, werden Fett dargestellt.



3.3.2.3 Dynamisierungsarten für Objekte

Die Objekte eines Anlagenbildes können auf unterschiedliche Art und Weise dynamisiert werden. Die einzelnen Standarddialoge für die Dynamisierung sind für unterschiedliche Zielgebiete ausgerichtet und führen zum Teil auch zu unterschiedlichen Ergebnissen.

Übersicht

Art	A	B	Vorteil	Nachteil
Dynamic Wizard	x	x	geführter Standardweg bei der Projektierung	nur für bestimmte Dynamisierungen. Erzeugt immer eine C-Aktion!
Direktverbindung		x	Die schnellste Dynamisierung im Bild, höchste Performance im Runtime	beschränkt auf eine Verbindung und nur innerhalb eines Bildes einsetzbar.
Variablenanbindung	x		Einfach zu projektieren	eingeschränkte Dynamisierungsmöglichkeiten
Dynamikdialog	x		Schnell und übersichtlich; für Wertebereiche oder mehrere Alternativen, hohe Performance im Runtime	nicht für alle Dynamisierungen einsetzbar
C-Aktion	x	x	Fast unbegrenzte Möglichkeiten der Dynamisierung durch die leistungsfähige Scriptsprache (Ansi-C)	Fehlermöglichkeiten durch falsche C-Befehle geringere Performance gegenüber anderen Dynamisierungsarten, daher immer überprüfen, ob das Ziel nicht durch eine andere Dynamisierungsart erreicht werden kann.

Legende:

A Dynamisierung der Objekteigenschaft
B Dynamisierung vom Objektereignis

Aufruf der Dialoge für die Dynamisierung

Dialog	Aufruf
Konfigura-tionsdialog	Nicht alle Objekte haben einen solchen Dialog. Automatisch beim Erzeugen dieser Objekte. Objekt im Bild selektieren → SHIFT-Taste drücken und gedrückt halten → 🖱D. Objekt im Bild selektieren → 🖱R das Kontextmenü öffnen → Konfigurationsdialog
Dynamic Wizard	Objekt im Bild selektieren → Eigenschaft oder Ereignis wählen → Wizard wählen und mit 🖱D starten . Der Dynamic Wizard muß über Ansicht → Symbolleisten... angewählt sein.
Direktverbin-dung	Objekt im Bild selektieren → Objekteigenschaften anzeigen → Register Ereignis anwählen →in der Spalte Aktion mit 🖱R das Kontextmenü öffnen → Direktverbindung wählen.
Variablenan-bindung	Objekt im Bild selektieren → Objekteigenschaften anzeigen → Register Eigenschaften anwählen →in der Spalte Dynamik mit 🖱R das Kontextmenü öffnen → Variable wählen → im Dialog die entsprechende Variable auswählen und übernehmen.
Dynamik-dialog	Objekt im Bild selektieren → Objekteigenschaften anzeigen → Register Eigenschaften anwählen →in der Spalte Dynamik mit 🖱R das Kontextmenü öffnen → Dynamik-Dialog. wählen → im Dialog die entsprechende Dynamik projektieren und übernehmen
C-Aktion	Objekt im Bild selektieren → Objekteigenschaften anzeigen → Register Eigenschaften oder Ereignis anwählen →in der Spalte Dynamik oder Aktion mit 🖱R das Kontextmenü öffnen → C-Aktion wählen → die entsprechende C-Aktion projektieren und übersetzen

Ergebnisse und Darstellung

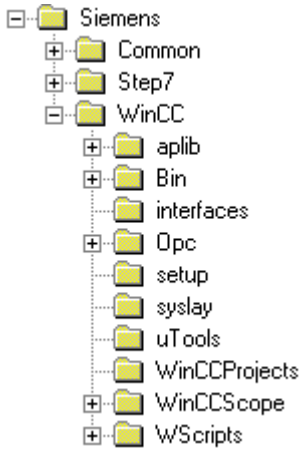
Dialog	Ergebnis	Darstellung
Dynamic Wizard	Es wird immer eine C-Aktion erzeugt.	Grüner Pfeil
Direktver-bindung		Blauer Pfeil
Variablenan-bindung		Grüne Glühbirne
Dynamik-dialog	automatisch generierte C-Aktion (InProc), diese C-Aktion kann nachträglich erweitert werden, dabei geht aber der Performancevorteil verloren	Roter Pfeil bei Änderung in C-Aktion Wechsel zu Grüner Pfeil
C-Aktion	projektiertes C-Script	Grüner Pfeil Gelber Pfeil - die Aktion muß noch übersetzt werden

3.3.3 WinCC Systemumgebung

WinCC wird unter dem Standard-Installationspfad *C:\Siemens\WinCC* angelegt. Während der Installation kann dieser Standardpfad geändert werden.

3.3.3.1 Verzeichnisstruktur WinCC System

Die Verzeichnisstruktur ohne Optionen und ohne Beispiele ist folgendermaßen aufgebaut.



Dateien im WinCC Standardverzeichnis

Im Standard-WinCC-Pfad sind für den Projekteur bzw. Inbetriebsetzer folgende Verzeichnisse und Dateien wichtig:

Verzeichnis	Dateiname, Extension	Bemerkung
Diagnose	License.log	aktuelle Logbucheinträge hinsichtlich der Lizenzprüfungen bzw. -verletzungen
	License.bak	die Logbuchdatei der Lizenzinformationen des letzten Startvorgangs
	WinCC_-Op_01.log	Operatormeldungen, die zur Laufzeit von WinCC erzeugt werden.
	WinCC_-Sstart_01.log	Systemmeldungen, die beim Anlauf von WinCC erzeugt werden. Eine wichtige Datei bei der **Fehlersuche**. Die Datei beinhaltet Meldungen über fehlende Variablen, fehlerhaft ausgeführte Scripte.
	WinCC_-Sys_01.log	Systemmeldungen, die zur Laufzeit von WinCC erzeugt werden. Eine wichtige Datei bei der **Fehlersuche**. Die Datei beinhaltet Meldungen über fehlende Variablen, fehlerhaft ausgeführte Scripte.
	S7chn01.log	Systemmeldung des verwendeten Kanales (hier S7)
aplib	Bibliotheks-pfad	die header-Dateien, alle Standard-Funktionen und alle Internen Funktionen, sind in Unterverzeichnissen abgelegt.

Dateien im WinCC Standardverzeichnis

Im Standard-WinCC-Pfad sind die projektübergreifenden Funktionen und Symbole in folgenden Verzeichnissen abgelegt:

Verzeichnis	**Unterver-zeichnis, Dateiname**	**Bemerkung**
aplib	library.pxl	Symbole der Standard-Bibliothek von WinCC
	Report, Wincc, Windows	Verzeichnisse für Standard-Funktionen, diese können **jederzeit** angepaßt werden.
	Allocate, C_bib, Graphics, Tag	Verzeichnisse für Interne-Funktionen, diese können **nicht** angepaßt werden.
syslay		alle Drucklayouts, die beim Anlegen eines Projektes von WinCC automatisch in den Projektpfad ins Verzeichnis *prt* kopiert werden.
wscripts	Dynwiz.cwd	*Dynamic Wizard* des *Graphics Designer*. Es können jederzeit eigene Scripte erstellt werden. Diese Scripte erhalten jeweils die Endung **.wnf**
	wscripts.deu	dieser Pfad enthält die Scriptdateien für *deutsch*. Dieser Pfad ist abhängig von der installierten Sprache.
	Wscripts.enu	dieser Pfad enthält die Scriptdateien für *englisch*. Da englisch die Defaultsprache ist, wird dieser Pfad immer angelegt.
	Wscripts.fra	dieser Pfad enthält die Scriptdateien für *französisch*. Dieser Pfad ist abhängig von der installierten Sprache.

Dateien im WinCC Standardverzeichnis

Folgende Anwendungsprogramme werden bei der Installation von WinCC in folgenden Verzeichnissen abgelegt:

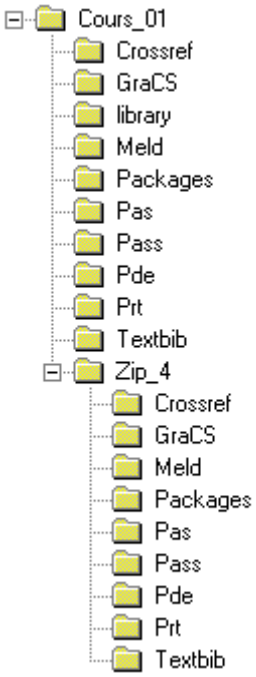
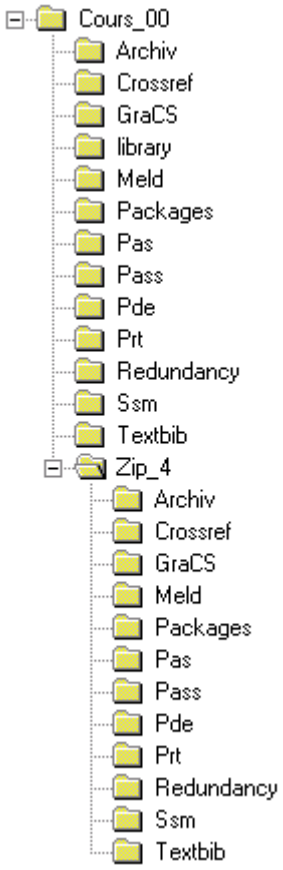
Verzeichnis\ Datei	Bemerkung
\sqlany\isql.exe	Interaktives Programm, um die Daten in der Datenbank eines WinCC-Projektes einsehen zu können.
\bin\Wunload.exe	Assistent (Wizard), um die Online-Tabellen in der Datenbank des WinCC-Projektes zu leeren, z.B. Entfernen der abgelegten Meldungen und Meßwertdaten. Der Assistent stellt die Laufzeittabellen automatisch für die Entladung ein; weitere Tabellen können jedoch vom Anwender jederzeit aus der Liste hinzugefügt bzw. entfernt werden. Dieses Tool muß offline zu einem WinCC-Projekt eingesetzt werden. Im Runtime-Modus kann dieses Tool nicht genutzt werden. Das Auslagern von Meldungen und Meßwerten zur Laufzeit (Runtime-Modus) kann mittels des Optionspaketes *STORAGE* durchgeführt werden.
\bin\Wrebuild.exe	Assistent (Wizard) für die Rekonstruktion der Datenbank, der **nicht** im Runtime-Modus eingesetzt werden kann!
\SmartTools\CC_GrafikTools\metaVw.exe	Viewer für Grafikdateien (z.B. Druckaufträge, exportierte Symbole) im EMF-Format (extended meta file).
\SmartTools\CC_GrafikTools\wmfdcode.exe	Viewer für Grafikdateien im WMF-Format (windows meta file).
\SmartTools\CC_OCX_REG\ocxreg.exe	Zur Registrierung bzw. zur Aufhebung der Registrierung von weiteren OLE Control Komponenten (OCX).
\SmartTools\CC_OCX_REG\Regsvr32.exe	wird von ocxreg.exe aufgerufen.

3.3.4 WinCC Projektumgebung

Hinweis:
Legen Sie für WinCC Projekte ein spezielles Projektverzeichnis an, z.B. WinCC_Projekte. Es wird eine klare Trennung zwischen dem WinCC System und den projektierten Daten erzielt. Auf diese Weise vereinfacht sich die Datensicherung. Die Gefahr des Datenverlustes (durch Bedienfehler) bei einer Deinstallation von WinCC wird weitestgehend umgangen.

3.3.4.1 WinCC Projekt - Verzeichnisstruktur

Ein Projekt unter WinCC besteht aus einer Verzeichnisstruktur mit entsprechendem Inhalt. Nach dem Anlegen eines neuen Projektes im *WinCC Explorer* (über den Menüpunkt *Datei → Neu*) wird eine Verzeichnisstruktur folgendermaßen aufgebaut:

WinCC Standard	WinCC mit Optionen
	

Inhalt der Projektverzeichnisse

Verzeichnis	Extension	Bemerkung
Projektpfad	.db	die Datenbank mit den Projektierungsdaten
	rt.db	die Datenbank mit den Laufzeitdaten (Runtime) Meßwerte, Meldungen
	.mcp (master control program)	Hauptdatei des WinCC Projektes. Mit dieser Datei wird das Projekt geöffnet.
	.pin	Projekt.pin
GraCS	.pdl (picture design language)	die projektierten Bilder
	.sav	Sicherungsdateien der Bilddateien mit dem letzten Projektierungszustand
	.gif (bitmap), .wmf (windows meta file), .emf (extended meta file)	Bilddateien
	.act (action)	exportierte *C-Aktionen*
	.pdd	*Default.pdd* Einstellungsparameter für den Grafikeditor (Default-Einstellungen der einzelnen Objekte in der Objektpalette)
Library	.h (header)	*Ap_pbib.h* (Projekt-Funktionsdeklarationen)
	.pxl	*Library.pxl* (Projekt-Symbolbibliothek)
	.fct	Projektfunktionen
	.dll (dynamic link library)	eigene Funktionsbibliotheken, die mit einer C Entwicklungsumgebung erstellt wurden.
Meld		
Pas	.pas (Aktions-definition)	Projekt-Aktionen, die als Hintergrundaktionen abhängig vom eingestellten Trigger ablaufen
Pass		
Pde		
Prt	.rpl (report picture language) .rp1 (Zeilenlayout)	Seitenlayouts für Druckaufträge Die vordefinierten WinCC-Standard-Layouts beginnen jeweils mit @. Alle Systemgrößen (auch Variable) sind mit diesem Präfix gekennzeichnet.
Rechnername z.B. Zip-ws1	\GraCS\GraCS.ini	Initialisierungsdatei für den Grafikeditor

Optional: Dateien die während der Projektierung entstehen können

Verzeichnis	Extension	Bemerkung
zum Teil frei bestimmbar	.ini	Initialisierungsdatei für den Simulator mit Informationen für den Aufruf.
	.sim	interne Variablen mit Einstellungen für die Simulation
	.csv	exportierte Texte aus der Textbibliothek
	.txt	exportierte Meldungen aus dem Meldesystem (*Alarm Logging*)
	.emf	Druckaufträge, die Ihre Druckergebnisse in eine Datei schreiben
	.log	Logdateien
	.xls .doc .wri	Dateien, die mit anderen Anwendungen erstellt wurden, aber im WinCC-Projekt genutzt werden

3.3.5 Automatischer Projektanlauf von WinCC

Anforderung

Das HMI-System (WinCC) auf der Anlage soll nach dem Einschalten des Windows-Systems automatisch starten. Der Operator an der HMI-Station benötigt keine Kenntnisse für die Bedienung des Windows-Systems (z.B. Aufruf von WinCC unter Windows NT).

Lösung

WinCC wird in der Anlaufroutine des PC‘s automatisch beim Hochlauf gestartet. Diese Einstellung wird im Ordner *Autostart* von Windows vorgenommen.

Verknüpfung anlegen

Schritt	Vorgehen bei NT4.0:
1	Im Windows Explorer in das Verzeichnis *WinNT\Profiles\All Users\Startmenü\Programme\Autostart* wechseln. WinNT ist das Verzeichnis in dem Windows NT installiert wurde
2	Im Ordner eine neue Verknüpfung mit der $\overset{\curvearrowright}{R} \rightarrow$ *Neu* $\rightarrow$ *Verknüpfung* anlegen
3	Die Verbindung zum Programm *mcp.exe* (Master Control Program) im *\bin* Verzeichnis von WinCC herstellen.
4	Der Verbindung einen Namen geben.

Dadurch wird das WinCC automatisch gestartet. WinCC selbst startet automatisch mit dem Projekt, das zuletzt bearbeitet wurde oder aktiviert war.
Um eine Anlage im Runtime-Modus zu starten, muß also das Projekt im aktiven Zustand verlassen werden.

Hinweis:
Ist die Tastenkombination *CTRL + SHIFT* nicht gesperrt und wird sie beim Anlauf von WinCC gedrückt, startet WinCC im Projektiermodus, auch wenn das Projekt im aktiven Modus beendet wurde.

Der Operator sieht nun sein bekanntes Startbild der Anlage. Damit der Operator nicht ungewollt bzw. absichtlich in die Projektierung (die im Hintergrund läuft) umschalten kann bzw. Windows Applikationen nutzen kann, die er nicht bedienen soll oder darf, müssen entsprechende Vorkehrungen getroffen werden. Der Operator darf auch nicht in das Datenbank Laufzeitfenster gelangen, da er über diesen Weg die WinCC Datenbankverbindung schließen kann.

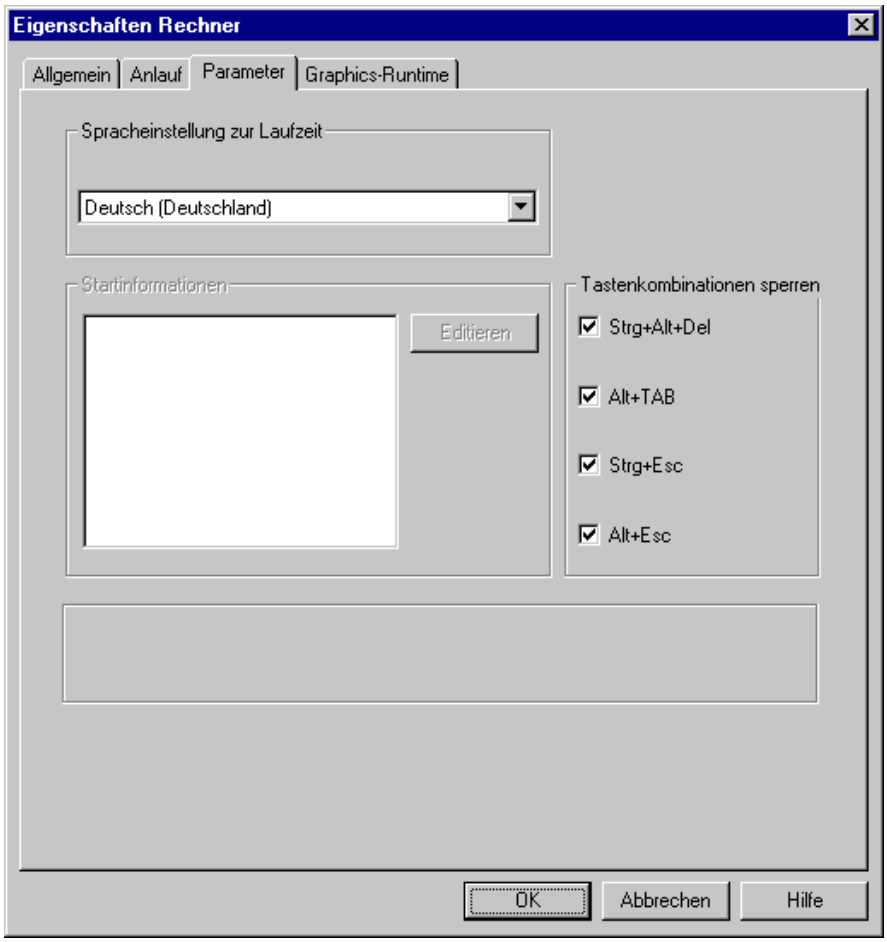
Keine Anwahl von:

Der Operator sollte keine Möglichkeit haben vom WinCC Runtime zu wechseln in:

- das *WinCC Explorer* von WinCC (Projektierungsumgebung),
- das Laufzeitfenster der *SQL-Datenbank* von WinCC (Sybase SQL Anywhere), da über diesen Weg ein Anwender die WinCC Datenbankverbindung schließen kann. WinCC kann dann nicht mehr weiterarbeiten,
- die *Taskleiste* von Windows, da über diesen Weg **alle** installierten Programme gestartet werden können,
- das aktuelle Taskfenster, da die Applikation geschlossen werden kann.

Notwendige Einstellungen des Rechners

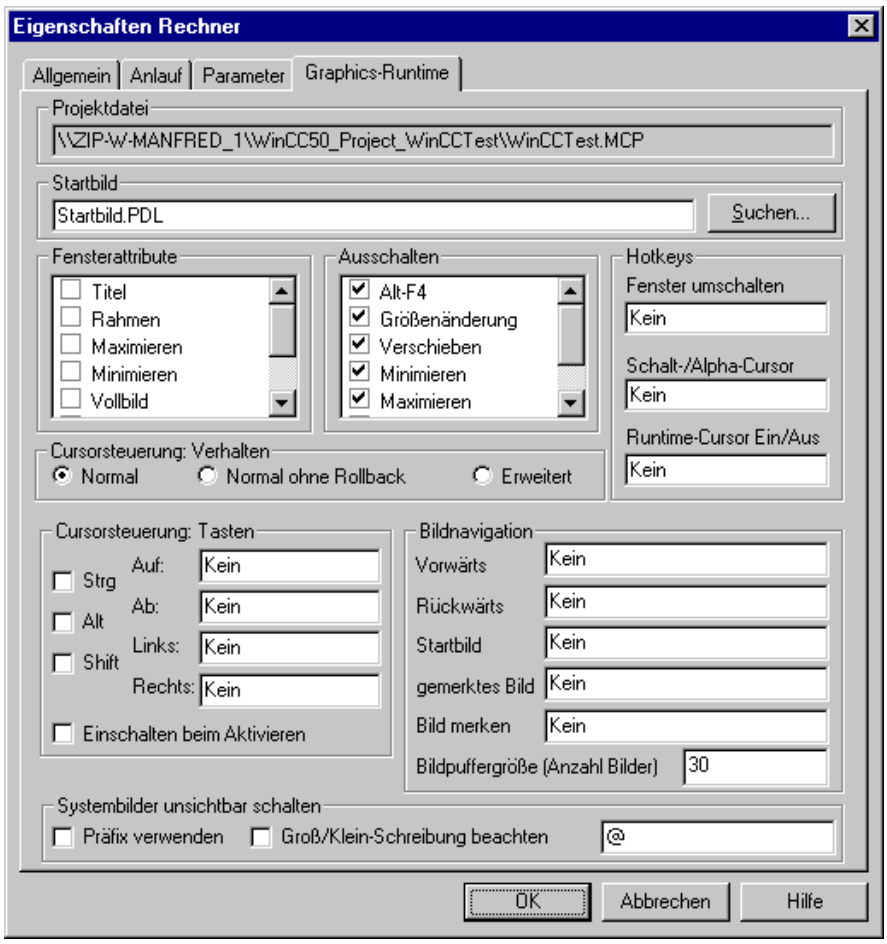
Damit der Operator diese Möglichkeiten nicht hat, müssen folgende Tastenkombinationen gesperrt werden.



Das Sperren erfolgt im *WinCC Explorer* im Dialog *Eigenschaften Rechner*.
Die genaue Bedeutung der einzelnen Tastenkombinationen kann aus der Hilfe zu WinCC bzw. der Hilfe zum jeweiligen Betriebssystem entnommen werden.

Notwendige Einstellungen des Runtime:

Über die Standard-Windows-Tasten könnte das Anlagenbild geschlossen werden; d.h. über diesen Weg könnte das Verlassen von WinCC auch erreicht werden. Um dies zu verhindern, müssen folgende Einstellungen für die Eigenschaften des Anlagenbildes vorgenommen werden:



Das Sperren erfolgt im *WinCC explorer* bei *Eigenschaften Rechner*.
Die genaue Bedeutung der einzelnen Tastenkombinationen kann aus der Hilfe zu WinCC bzw. der Hilfe zum jeweiligen Betriebssystem entnommen werden.

- wenn *Größenänderung* und *Minimieren* nicht ausgeschaltet sind, wird die Oberfläche des Betriebssystems wieder zugänglich.
- *Schließen* (im Bild nicht dargestellt) muß ebenfalls ausgeschaltet werden. Andernfalls kann das Runtime beendet werden und der Anwender gelangt in das Configuration System.

Hinweis:
Werden die genannten Tasten alle oder teilweise gesperrt, muß dem Projekteur bzw. dem Servicepersonal der Zugang zur Projektierung mittels einer eigens dafür projektierten Taste ermöglicht werden. Dies gilt auch für das ordentliche Herunterfahren der Anlage.

Diese Funktionen dürfen nicht frei bedienbar sein. Es ist zusätzlich ein **Zugriffsschutz**, z.B. Service Personal, hinter der Eigenschaft *Paßwort* des Buttons zu hinterlegen.

3.3.6 Koordiniertes Beenden von WinCC

System konsistent halten

Um das System konsistent zu halten, ist es sehr wichtig, daß WinCC korrekt beendet wird. WinCC Runtime sollte immer als erstes und mittels der dazu vorgesehenen Button beendet werden. Erst danach wird das Projekt über den WinCC Explorer geschlossen
Sollte das nicht eingehalten werden, so könnte es zu Datenverlust, einer beschädigten Datenbank und im schlimmsten Fall zu einem Systemabsturz kommen.

Beenden durch Bedienung

Eine WinCC-Station darf niemals ohne Shutdown des Betriebssystems ausgeschaltet werden. Ein *NOTAUS-Schalter* ist für das HMI-System nicht geeignet. Deshalb muß für den Operator eine entsprechende Bedientaste projektiert werden, die Ihm erlaubt, ohne zusätzliches Wissen das System korrekt zu beenden.

Stromausfall

Um Datenverluste durch Stromschwankungen bzw. Stromausfall möglichst zu vermeiden, sollte ein detailliertes **Datensicherungskonzept** für das HMI-System ausgearbeitet und angewendet werden.

Prinzipiell sollte für die WinCC-Station auf jeden Fall eine USV (Unterbrechungsfreie Stromversorgung) eingeplant werden. Dies kann der Anschluß an die betriebsinterne USV sein oder der Anschluß einer eigenen USV an den Server von WinCC. Dies gilt sowohl für Einplatz- als auch für Mehrplatzsysteme unabhängig vom eingesetzten Betriebssystem (Windows NT).
Die eingesetzte USV muß zusätzlich eine spezielle Shutdown-Software für Windows NT beinhalten, damit bei einem Stromausfall und dem Abschalten nach einer bestimmten Zeitspanne das Betriebssystem und alle aktiven Applikationen ohne Datenverlust automatisch heruntergefahren werden; z.B.: APC USV 600 mit Power-Shutdown Software für Windows NT.

3.3.6.1 Hinweise zur Installation einer USV

Für den Anschluß einer USV mit entsprechender Prüfsoftware muß eine serielle Schnittstelle zur Verfügung stehen. Falls an der WinCC Station keine serielle Schnittstelle frei ist, z.B. belegt durch Drucker- oder PLC-Anschluß, muß eine zusätzliche Schnittstellenkarte eingesetzt werden.
Mehrfach genutzte serielle Schnittstellen (z.B. über Switches) werden von den meisten USV-Systemen nicht unterstützt und sind auch nicht sinnvoll, da eine ständige Überwachung des Systems nötig ist.
Im Betriebssystem wird ein entsprechender Überwachungsdienst installiert. Dieser Überwachungsdienst muß noch mit Shutdown Parametern versorgt werden, um ein koordiniertes Beenden des Systems zu garantieren. Der Shutdown Prozeß für Applikationssoftware muß auf jeden Fall aktiviert werden, damit WinCC bei einem Shutdown ohne Datenverlust heruntergefahren wird. Abhängig von den aktiven Applikationen muß die *Sicherungszeit* vor dem Shutdown ausreichend groß gewählt werden.

Die Software für die USV-Systeme bieten meistens auch einen zeitlich gesteuerten *Shutdown* z.B. für das Wochenende oder für die Nacht. Über diesen Weg kann ein **gezieltes Beenden des WinCC-Systemes ohne Bedienung** erreicht werden.

3.3.7 Datensicherung

Wann soll gesichert werden?

Ein WinCC-Projekt muß aus folgenden Gründen wiederholt gesichert werden:

- Datensicherung während der Projektierungsphase.
- Sicherung vor dem Export bzw. Import von Daten (z.B. beim Import von Variablen, von mehrsprachigen Bildtexten, von Meldungstexten sowie von mehrsprachigen Meldungstexten).
- Sicherung vor der Neubildung bzw. dem Entladen der WinCC-Datenbank.
- Sicherung vor der Bearbeitung der Datenbank mit Tools wie z.B. interaktiver SQL-Zugriff.
- Sicherung der Projektierungsdaten für die Installation auf der Zielanlage beim Endkunden.
- Übernahme von Daten für ein ähnlich aufgebautes Projekt.

Welche Medien sind geeignet?

Medium	Vorteil	Nachteil
Disketten	fast überall lesbar	zu geringe Kapazität (auch in gepackter Form)
ZIP-Disketten	preiswert, ausreichende Kapazität; direkter und schneller Zugriff über Windows möglich; einfach installierbar; mobil, für den Einsatz auf der Anlage	
Streamer (z.B. im Netzwerk)	automatische Sicherung möglich (täglich), sehr große Kapazität	meist nur in Büroumgebung vorhanden, kein direkter Zugriff auf die Daten da spezielles Format beim Speichern
Festplatte auf einem anderen PC (z.B. Laplink)	kein Medien-Handling, Daten direkt verwendbar	langsam, bei großen Datenmengen eher ungeeignet
MOD	hohe Datensicherheit, wiederverwendbar, Sicherung von Meldungen und Meßwerten im Runtime-Modus möglich.	für das Schreiben und Lesen ist ein spezielles Laufwerk notwendig
CD-ROM	hohe Kapazität, fast überall lesbar, für Langzeitarchivierung geeignet.	für das Beschreiben ist ein spezielles Laufwerk notwendig, nicht wiederverwendbares Medium

Projekt vor der Datensicherung bereinigen

Um das Projekt vor der Datensicherung für den Endkunden bzw. für die Übernahme von Projektdaten möglichst **bereinigt** zu sichern, können vorher folgende Daten gelöscht bzw. mittels Zusatzprogrammen bereinigt werden.

- Alle Sicherungsdateien im Verzeichnis *\GraCS*.sav* des Projektes.
- Wenn für die Dokumentation keine eigenen Layouts (Report Designer) erstellt wurden, können die Systemlayouts aus dem Verzeichnis *\Prt* gelöscht werden. Beim Anlegen eines neuen Projektes werden automatisch alle Systemlayouts aus dem Verzeichnis *\Siemens\WinCC\syslay* in das Projektverzeichnis kopiert.

Was muß gesichert werden?

Sollen nur die Daten eines WinCC-Projektes gesichert werden, müssen folgende Dateien und Verzeichnisse mit ihren Dateien gesichert werden.
aus dem Projektverzeichnis:

- die Dateien **.mcp, *.pin, *.db*
- die Verzeichnisse *\GraCS* und *\Library*
- falls eigene Aktionen erstellt wurden, das Verzeichnis *\Pas*
- falls eigene Drucklayouts erstellt wurden, das Verzeichnis *\Prt.*

Wenn projektübergreifende Komponenten (Standardfunktionen, Objekte in der Projektbibliothek) erstellt wurden, müssen zusätzlich folgende Dateien aus
dem WinCC-Standardverzeichnis:

- die Dateien *\Siemens\WinCC\aplib*.fct*
- die Datei *\Siemens\WinCC\aplib\library.pxl*

gesichert werden. Diese Daten werden bei einer Neuinstallation von WinCC **nicht** generiert.

3.3.8 Kopieren eines gesicherten WinCC-Projektes auf eine neue Ziel-Maschine

Installation der Systemsoftware

Installation der WinCC-Software. Dies erfolgt bei Komplettsystemen über den automatisch aufgerufenen Konfigurationsdialog oder über die Installations-CD, die mit dem WinCC-Paket und den zugehörigen Lizenzdisketten geliefert werden.

Zusätze

Werden in Ihrem Projekt spezielle Zusatzpakete (z.B. Optionspakete oder Addons) oder spezielle Kommunikations-Schnittstellen oder Schnittstellen zu anderen Windows Programmen (z.B. WORD, EXCEL etc.) verwendet, so sind diese Pakete ebenfalls auf der Zielmaschine zu installieren. Die zugehörigen Autorisierungen für Optionspakete, Addons oder Kommunikations-Schnittstellen (Kanal-DLLs) sind auch auf der Zielmaschine zu installieren. Beachten Sie, daß **alle** nötigen Autorisierungen (für alle genutzten Kanal-DLLs) eingespielt werden müssen, um mit dem WinCC-Projekt arbeiten zu können.

Windows Software

Werden in den WinCC-Bildern OLE-Verbindungen zu anderen Windows Programmen, z.B. zu WORD, ClipArts oder EXCEL verwendet, so muß abhängig von der Art der OLE-Verbindung auf der Zielmaschine das zugehörige Softwarepaket ebenfalls installiert sein, d.h. in der Windows Registrierung eingetragen sein.

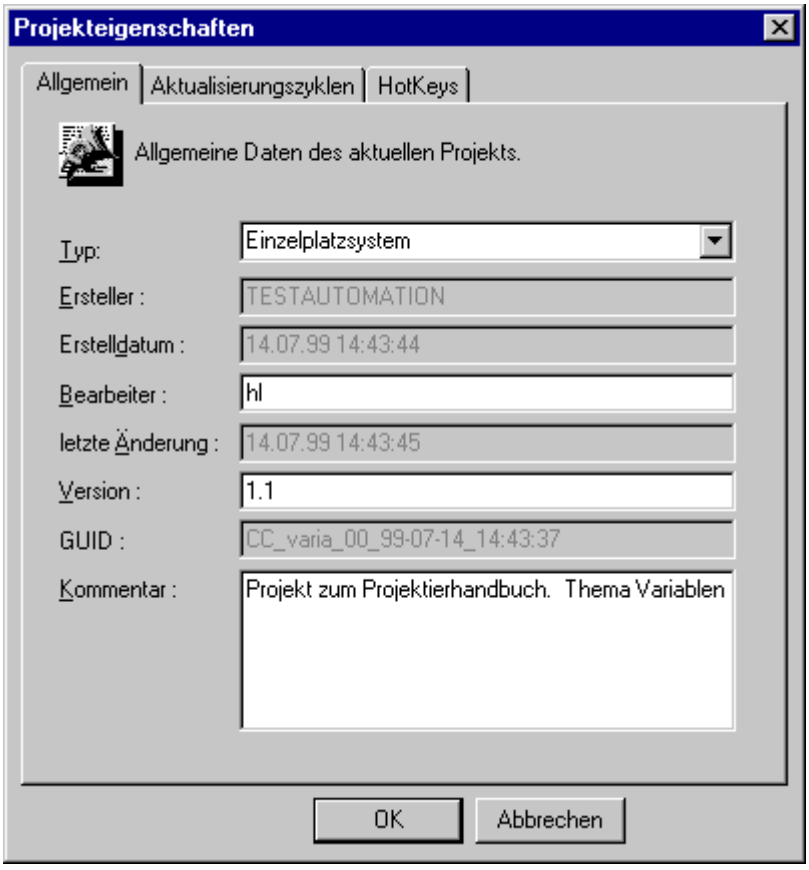
OCX, ActiveX

Werden weitere OCX-Komponenten (OLE Control, ActiveX) aus zugekauften Softwarepaketen eingesetzt, so müssen diese ebenfalls in der Windows Registrierung hinterlegt sein. Registrieren bzw. prüfen Sie die Registrierungseinträge dieser OCX-Komponenten z.B. mittels des auf der WinCC CD mitgelieferten Tools *SmartTools\CC_OCX_REG\ocxreg.exe*. Wird eine Registrierung zu OLE-Objekten bzw. OCX-Objekten beim Anlauf des WinCC-Projektes im Runtime-Modus (als auch in der Projektierung mittels des *Graphics Designer*) nicht gefunden, so wird dieses Objekt im Bild mit dem Hinweis *Unknown Object* eingeblendet.

Netzwerk

Wurde das Projekt für ein **Mehrplatzsystem** konfiguriert, muß vor dem Kopieren der WinCC Daten die gesamte Netzinstallation auf der WinCC Zielmaschine durchgeführt werden. Notieren Sie sich die nötigen Rechnernamen in der konfigurierten Rechnerlandschaft, da diese bei der Parametrierung des kopierten Projektes benötigt werden. Der Rechnername wird auch bei einem Einplatzsystem als Parameter benötigt, daher müssen Sie auf jeden Fall die Namensgebung auf der Zielmaschine kennen bzw. über die Windows Systemsteuerung ermitteln.

Kopieren der Daten und Starten des Projektes

Schritt	Vorgehen beim der Datenübernahme
1	Anlegen eines Projektverzeichnisses (z.B. *WinCC_Projekte*)
2	Den gesamten gesicherten Pfad in dieses eingerichtete Verzeichnis als Unterverzeichnis kopieren (z.B.: *\WinCC_Projekte\Varia_00*). Der Name des WinCC-Projektverzeichnisses kann, falls gewünscht, geändert werden. Beim Umbenennen der Dateien im Projektverzeichnis (varia_00.mcp, varia_00.db, varia_00.pin, varia_00.log) ist darauf zu achten, daß alle Dateien den gleichen Namen (außer der Erweiterung) erhalten.
3	Das Projekt im WinCC *Explorer* öffnen.
4	Bei Bedarf die projektspezifischen Einstellungen ändern. Für die Änderung des Typs sind zusätzliche Anpassungen bei den *Rechnereigenschaften* notwendig. Diese Anpassungen sind in der WinCC Hilfe beschrieben. 

Schritt	Vorgehen beim der Datenübernahme
5	Den *Rechnernamen* überprüfen und gegebenenfalls im Dialog *Eigenschaften Rechner* auf der Registerkarte *Allgemein* ändern. Stimmt der Rechnername im WinCC-Projekt nicht mit dem Rechnernamen dieses Zielsystems überein, wird beim Aktivieren des Projektes, d.h. beim Anlauf im Runtime-Modus, eine Fehlermeldung angezeigt.
6	Wurden im Projekt eigene *Standard-Funktionen* (*.fct) verwendet, müssen die gesicherten Standard-Funktionen in den WinCC-Standardpfad *\Siemens\WinCC\aplib* kopiert und anschließend im Funktionsbaum bekannt gegeben werden. • im geöffneten Projekt den Editor *Global Script* aufrufen • die Deklarationsstruktur über den Menüpunkt *Optionen → Header neu generieren* neu erstellen. Nun sind die neuen Funktionen im Funktionsbaum sichtbar.
7	Das Projekt aktivieren, um den korrekten Anlauf des Projektes zu überprüfen.

3.3.9 Wiederverwendung - Übernahme von Projektteilen in ein neues bzw. bestehendes Projekt

Der Grund für die Übernahme

Ein WinCC-Projekt kann auf unterschiedliche Weise entstehen. Dabei sind die wichtigsten Gesichtspunkte die Wiederverwendbarkeit bereits bestehender Projektteile aus ähnlichen Projekten bzw. die Datenübernahme aus vorprojektierten Beispielprojekten.

Projektteam

Bei einem Projektierungsteam werden in dieser Hinsicht ähnliche Aufgaben zugrunde gelegt, da ein WinCC-Projekt letztendlich wieder zu einem Projekt verschmolzen werden muß.
Ein WinCC-Projekt besteht aus einzelnen Dateien (z.B. Bilder) **und** den Projektierungsdaten in der Datenbank (Meldesystem, Variablenhaushalt).

Daten in der Datenbank

Daten, die in der Projektierungsdatenbank hinterlegt sind, können **nicht** in zwei Einzelprojekten erstellt und anschließend zusammengeführt werden.
Aus diesem Grunde muß bei der Projektierung von Datenbankinformationen (z.B. Aufbau des Meldesystems) ein **Basisprojekt** erstellt werden, das für diese Art der Projektierung verwendet wird. Dieses Basisprojekt muß vor **jeder** Datenbankänderung (auch bei Zwischenschritten) **gesichert** werden. Geht bei der Änderung etwas schief kann auf den Stand vor der Änderung zurückgegriffen werden.

Hinweis:
Beachten Sie hierbei, daß jede Änderung in der Datenbank die Struktur und den Zugriff auf die Datenbank beeinflussen kann. Viele unnötige Änderungen (mit evtl. Löschvorgängen) können dazu führen, daß die WinCC Datenbank nicht mehr optimal zusammengestellt ist (Performanceverlust).

Einzelplatzsystem

Während an dem Basisprojekt der nächste Projektierungsschritt z.B. beim Meldesystem durchgeführt wird, sollte bei einem WinCC *Einzelplatzsystem* auf keinen Fall an anderer Stelle eine Datenbankänderung vorgenommen werden (z.B. Archivierung – *Tag Logging*).

Mehrplatzsystem bzw. Multiclient

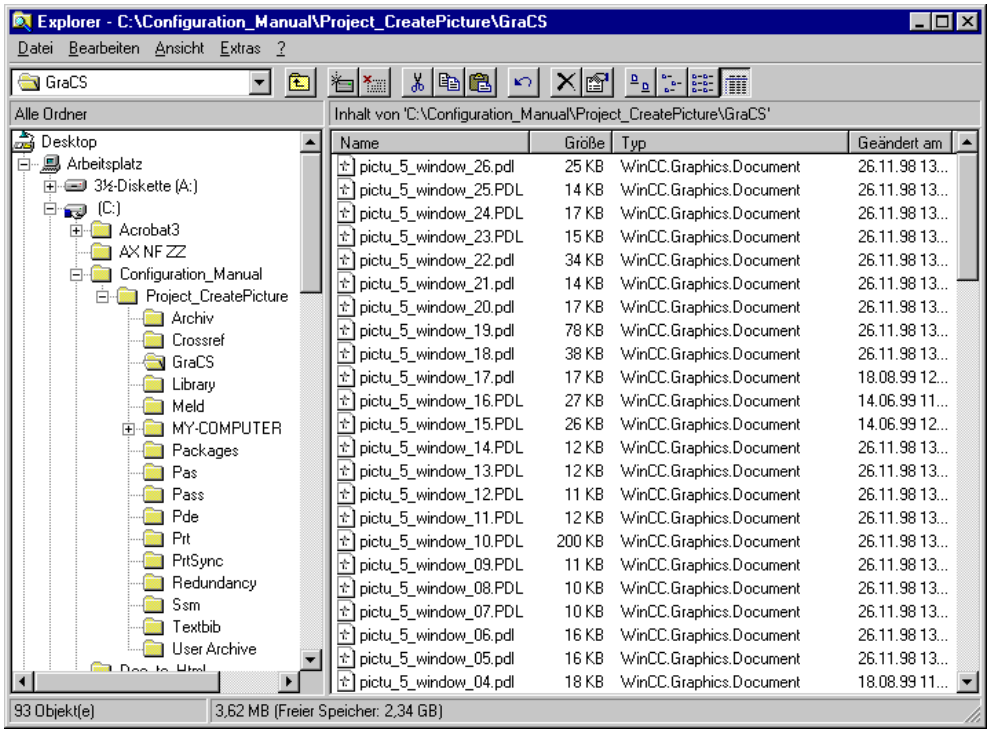
Wird dagegen auf einem WinCC *Mehrplatzsystem* oder auf einem *Multiclient* an einem Projekt projektiert, so können gleichzeitig an unterschiedlichen Datenbankbereichen Projektierungen vorgenommen werden. Zum Beispiel kann ein Projekteur das Meldesystem und ein weiterer Projekteur das Archivierungssystem (Meßwerterfassung) bearbeiten.

Änderung des Projekttys - Umwandlung Einzelplatz / Mehrplatzsystem

Jedes Projekt kann von einem beliebigen WinCC Projektierungs-System in ein anderes Projektierungs-System umgewandelt werden. Bei einem vorgesehenen Wechsel der Konfiguration zwischen Projektierungs-Station(en) und Kundensystem dürfen keine spezifischen WinCC-Elemente genutzt werden, die auf ein Mehrplatzsystem ausgerichtet sind. Zum Beispiel sollten keine rechnerlokalen internen Variablen genutzt werden, wenn eigentlich ein Einzelplatzsystem beim Kunden konfiguriert wird.
Beim Anlegen eines neuen Projektes (entweder Einzelplatz- oder Mehrplatzsystem) muß bereits bekannt sein, ob auf einem bereits vorhandenen Basisprojekt mit vorprojektiertem Meldesystem bzw. Archivierungsdaten aufgesetzt werden soll. Diese in der Datenbank hinterlegten Daten sind nur durch Neuprojektierung oder wenn möglich durch Export - Import in andere Projekte übertragbar.

3.3.9.1 Übernahme von Bildern

Projektierte Bilder können jederzeit übernommen werden. Sie können die Bilddateien (**.pdl*) entweder direkt über den Windows Explorer von dem Quellverzeichnis in das Zielverzeichnis des WinCC-Projektpfades *\GraCS* kopieren (geeignet bei mehreren Bilddateien). Nachfolgend ein Auszug aus dem Projekt für die Bildprojektierung.



Die zweite Art der Bildübernahme funktioniert über das Öffnen einer Bilddatei (*bild.pdl*) im *Graphics Designer* über den Menüpunkt *Datei → Datei öffnen.* Anschließend wird das Bild im aktuellen Bildverzeichnis (*\GraCS*) über den Menüpunkt *Datei → Speichern unter* abgelegt. Diese Vorgehensweise eignet sich dann, wenn die Bilddateien als Basis verwendet werden und sofort eine Anpassung durchgeführt werden soll.

Referenzen in Bildern

- Strukturen aus dem Bereich Datentypen
- Interne oder externe Variablen
- Systemvariablen
- Alarm Logging Controls (Meldefenster)
- Tag Logging Controls (Kurven- oder Tabellenfenster)
- User Archive Controls (Tabellenfenster)
- Bildobjekte, die als Bitmap oder Metafile abgelegt sind (z.B. für Zustandsanzeigen oder Grafikobjekte)
- Weitere Bilder, die als Grafik- oder Prozeßboxen bzw. eingeblendete Fenster genutzt werden
- Verwendete Projektfunktionen
- Zugriffsrechte

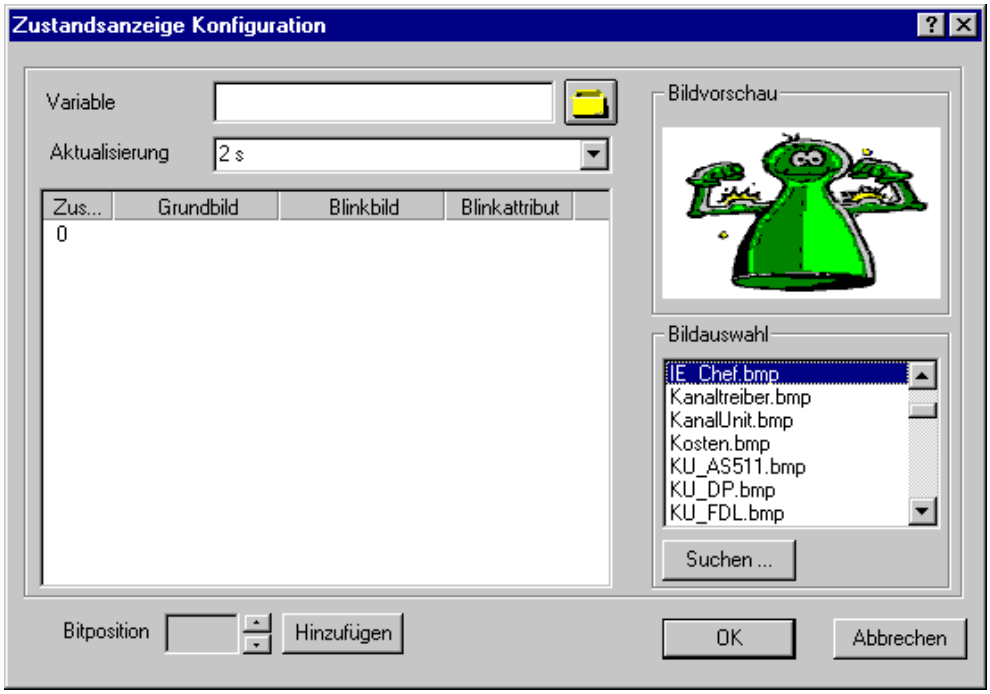
Folgende Referenzen müssen zusätzlich definiert werden:

- Definition der Strukturen unter Datentypen, z.B. Regler- oder Vorlagenstrukturen für Anwenderobjekte
- Definition der Kommunikationskanäle und logischen Verbindungen mit der Definition der Variablen (evtl. mit Variablengruppen)
- Definition der internen Variablen bzw. Systemvariablennamen (mit @ beginnend)
- Übernahme der Bildelemente (*.gif bzw. *.emf) durch Kopieren aus dem Verzeichnis *\GraCS*
- Übernahme von Bildfensterinhalten durch Kopieren der weiteren Bilddateien (*.pdl) aus dem Verzeichnis *\GraCS*
- Die verwendeten Projektfunktionen müssen vom Quellprojekt in das Verzeichnis *\Library* des neuen Projektes kopiert werden. Zusätzlich muß im Editor *Global Script* über *Header generieren* die Hinterlegung dieser Funktionen im Funktionsbaum erfolgen. Diese Vorgehensweise wurde bereits detaillierter im Kapitel 4.1 Entwicklungsumgebung für C Scripte beschrieben.
- Festlegung von verwendeten Zugriffsrechten im Editor *User Administrator*. Die verwendeten Zugriffsrechte (z.B. bei Bedientasten) müssen für die Gruppenfestlegungen definiert werden.

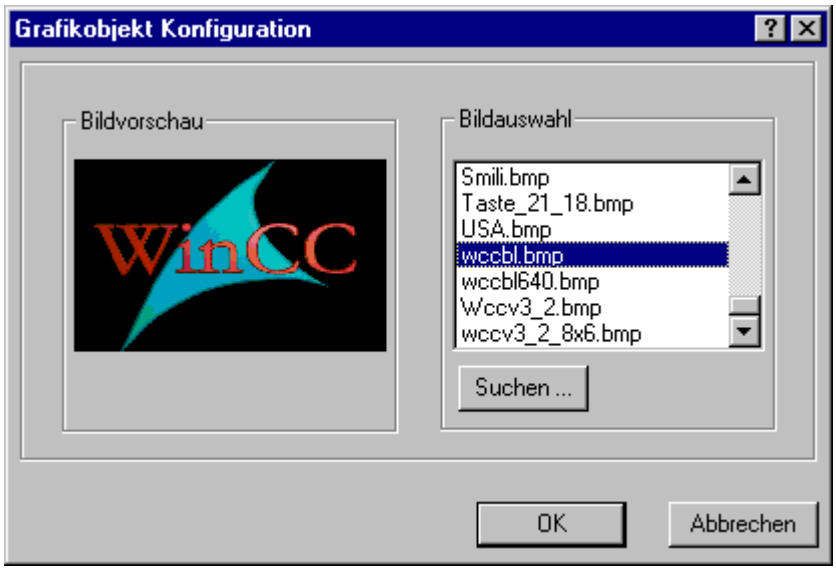
3.3.9.2 Übernahme von Symbolen und Bitmaps

Kopieren

Symbole für *Zustandsanzeigen* oder *Grafikobjekte* in Bilddateien werden als eigene Dateien in dem Bildverzeichnis des Projektes hinterlegt. Kopieren Sie dazu die gewünschten Symboldateien (*.emf oder *.gif) in das Zielverzeichnis *\GraCS* des neuen Projektes. Nun sind diese Bilder sofort in der Auswahlliste bei *Zustandsanzeigen* oder *Grafikobjekten* (siehe Objektpalette im *Graphics Designer*) verfügbar. Ausschnitt aus dem Konfigurationsdialog der *Zustandsanzeige*.



Bildauswahl für das *Grafikobjekt*.



Importieren

Symbole können entweder über den eben beschriebenen Weg in ein Bild eingebunden werden oder können direkt über den Menüpunkt *Einfügen → Importieren* in das in Bearbeitung befindliche Grafikbild kopiert werden. In diesem Fall muß keine Datei kopiert werden, sondern Sie importieren das gewünschte Symbol direkt über den Zugriff auf den Pfad des Quellprojektes (*\GraCS*) und der gewünschten Symboldatei (*.gif, *.emf, *.wmf). Das Symbol erscheint nach dem Import direkt als Objekt im Bild (oben links).

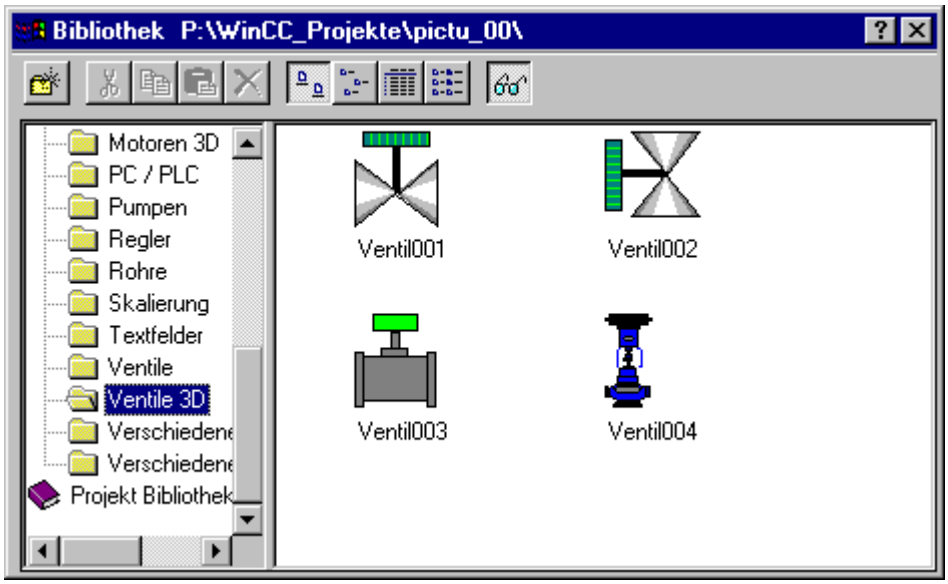
Werden Symbole in einer Projektbibliothek hinterlegt, so kann die Projektbibliothek in einem weiteren Projekt durch die Übernahme der gesamten Projektbibliothek genutzt werden. Siehe dazu die nachfolgende Beschreibung.

3.3.9.3 Übernahme einer Projektbibliothek (mit vorprojektierten Symbolen und Anwenderobjekten)

Globale Bibliothek

Werden Symbole in einer Projektbibliothek abgelegt, so kann diese in einem weiteren Projekt durch kopieren der Datei *library.pxl* in den Pfad *\Library* weiter genutzt werden.

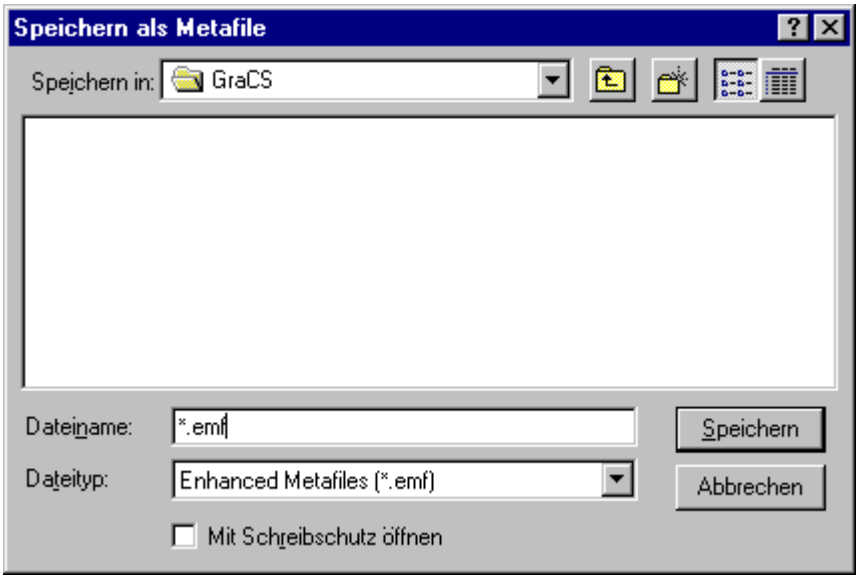
Die vorprojektierten Bausteine können nun jederzeit im neuen Projekt weiter verwendet werden:



Hinweis:
Berücksichtigen Sie dabei, daß verbundene Symbole eventuell auf nicht vorhandene Referenzen (oder Variablen) verweisen, die Sie erst definieren müssen. Abhängig von der Projektierung dieser Symbole müssen die dazugehörigen Aktionen oder Verbindungen eventuell angepaßt werden. Prüfen Sie daher nach der Verwendung von Symbolen aus der Bibliothek, welche Eigenschaften/Ereignis-Verbindungen bereits vorliegen und ob diese eventuell angepaßt werden müssen.

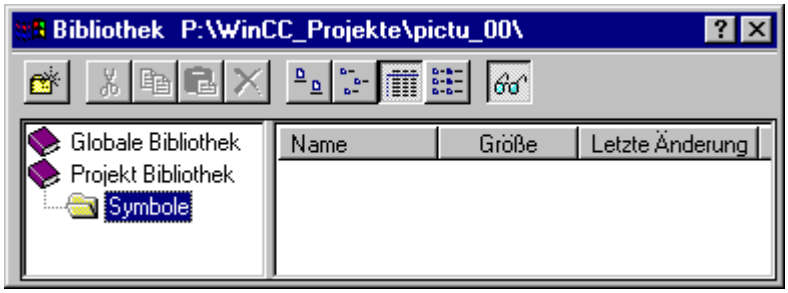
Einzelne Symbole

Werden nur einige bestimmte Symbole aus der Projektbibliothek im neuen Projekt genutzt, so werden diese Symbole einzeln exportiert (Symboldatei *.emf).

Schritt	Vorgehen: Symbole übernehmen
1	Bibliothek öffnen.
2	Das gewünschte Symbol mit der auswählen und bei gedrückter Maustaste das Symbol in das Bild ziehen (Drag&Drop).
3	Mit *Datei → Export...* den Dialog zum Speichern des Symbols aufrufen. 
4	Das Symbol speichern.

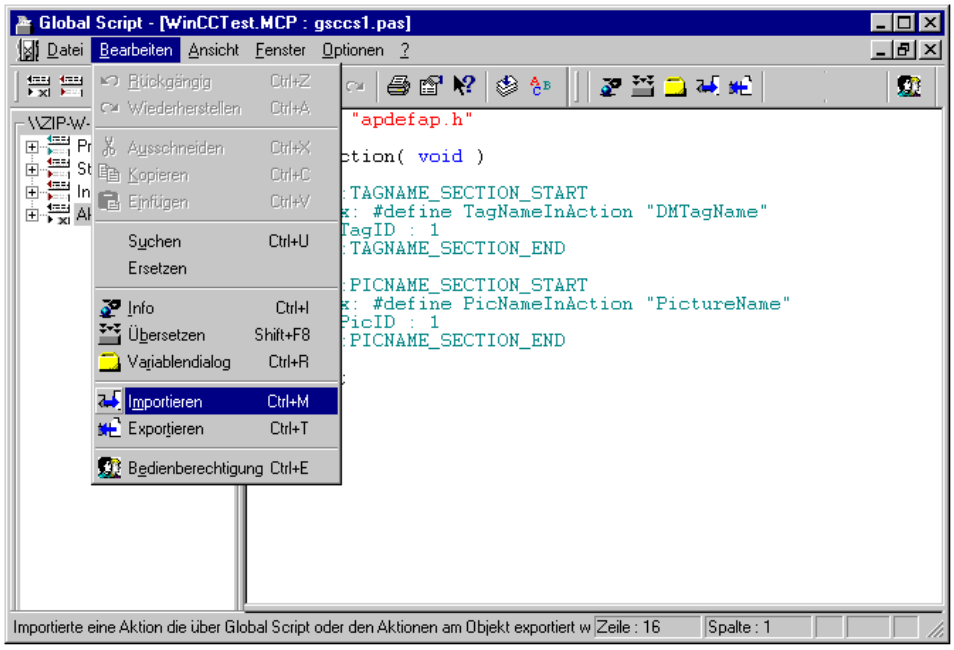
Neue Projektbibliothek

Diese exportierten Symbole stehen nun als einzelne Symboldateien zur Verfügung und können über den Weg des Imports einzeln genutzt werden. Werden diese Symbole im Projekt häufiger verwendet, so sollten diese Symbole wieder in die neue Projektbibliothek eingehängt werden. Rufen Sie dazu die Symbolbibliothek, speziell die Projektbibliothek auf.
Legen Sie ein eigenes Symbolverzeichnis, z.B. mit Hilfe des Verzeichnissymbols in der Symbolleiste des Bibliotheksfensters an und kopieren Sie mittels *Drag&Drop* die importierten Symbole in dieses Verzeichnis. Auf diesem Wege können Sie teilweise Ihre Symbole aus Projekten übernehmen und weitere spezielle Symbole hinzufügen, damit wieder eine eigene projektspezifische Bibliothek entsteht.



3.3.9.4 Übernahme von Aktionen

Aktionen, die in einem Projekt häufiger benötigt werden oder von einer Objektaktion an eine andere Objektaktion kopiert werden sollen, werden als eigene Dateien abgelegt. Diese Dateien werden im Verzeichnis *\GraCS* mit der Erweiterung *.act* gespeichert. Diese Dateien können durch Kopieren vom Quellverzeichnis in das Zielverzeichnis jederzeit übernommen werden.
Eine Aktionsdatei wird über den Editor für *C-Aktionen* mittels der Taste *Aktion exportieren* in der Symbolleiste in die selbst benannte Zieldatei (mit der Endung *.act* für Aktion) gespeichert.



Mit der Taste *Aktion importieren* wird eine gespeicherte Aktionsdatei für eine Objektaktion im Bild des neuen Projektes übernommen. Beachten Sie dazu auch die Beschreibung in Kapitel 4.1 Entwicklungsumgebung für C Scripte.

Hinweis:
Häufiger verwendete Aktionen können auch als Projekt- oder Standard-Funktionen definiert werden können.

3.3.9.5 Übernahme von Variablen

Der Variablenhaushalt von WinCC kann über mehrere Wege ergänzt werden:

- Einlesen von S5-Datenvariablen bzw. S7-Datenvariablen über Assistenten (*Dynamic Wizard*)
- Übernahme von *S7-Variablen* mittels des *PCS7-Mappers*
- Import und Export von Textlisten mittels des Zusatztools *Var_Exim*
- Interaktiver Zugriff auf die Datenbanktabellen (Variablentabellen)
- Eigens programmierte Assistenten (*Dynamic Wizard*) oder Programme, die mittels der WinCC API-Funktionen neue Daten im Variablenhaushalt erzeugen

Die letzten zwei aufgeführten Möglichkeiten setzen sehr gute Kenntnisse hinsichtlich des Umganges mit SQL-Datenbanken sowie der Programmierung über die Applikationsschnittstelle voraus. Dies sollte nur von Personen mit solchen Kenntnissen durchgeführt bzw. vorbereitet werden.

Vor der Übernahme der Daten in das Zielprojekt muß daher geklärt werden, wo die Basis für das Zielprojekt liegt. Liegt bereits eine größere Anzahl von Variablen im Variablenhaushalt von WinCC vor, so sollte die Variablenliste von WinCC als Import für das Zielprojekt herangezogen werden. Die *internen Variablen* müssen auf jeden Fall aus dem *Variablenhaushalt* von WinCC übernommen werden. Dazu muß das Tool *Var_Exim.exe* eingesetzt werden.

S5-/S7-Datenvariablen mit Dynamic Wizard übernehmen

Die mit STEP5/STEP7- Software erstellten Datenbereichsdefinitionen können in den Variablenhaushalt von WinCC mit Hilfe der vorhandenen *Dynamic Wizards* eingelesen werden. Folgende Tätigkeiten sind durchzuführen:

Schritt	Vorgehen: Daten von S5 oder S7 übernehmen
1	Eine **Datensicherung** des Projektes durchführen. Es werden Veränderungen in der Datenbank vorgenommen.
2	Mit der STEP-Software die Zuordnungsliste exportieren. Es entsteht eine Datei *prj_zuli.SEQ*
3	Diese exportierte Datei von speziellen Symbolen (z.B. zu Programmaufrufen) bereinigen, die für den Import in WinCC nicht benötigt werden. Dies kann mit Hilfe eines Texteditors (z.B. Wordpad) durchgeführt werden. Die Zuordnungsliste darf keine Leerzeilen enthalten.
4	Das Ziel-Projekt im WinCC *Explorer* öffnen. Das Projekt muß sich im Projektierungsstatus befinden (Runtime nicht aktiv).
5	Den Editor *Graphics Designer* öffnen. In einem beliebigen Bild im *Dynamic Wizard* (einblenden über *Ansicht → Symbolleisten...*) die Registerkarte *Import-Funktionen* wählen. Dort die Funktion *Import S7 – S5 ZULI* wählen. Im folgenden müssen Sie die *Quell-Datei (.seq)* mit Verzeichnisangabe nennen. Des weiteren ist die logische Verbindung anzugeben, in die Variablenbeschreibungen der Zuordnungsliste eingehängt werden sollen. Die Daten werden nun in den WinCC- Variblenhaushalt eingetragen. Die Variablennamen müssen im gesamten WinCC-Projekt eindeutig sein. Die Variablen werden dem evtl. bereits bestehenden WinCC-Variablenhaushalt hinzugefügt. Der Variablenname ist dabei der Schlüssel.

Variablenübernahme mit dem Hilfsprogramm

Die Verbindungen (Kanal-DLL, logischer Verbindung sowie Verbindungsparameter) müssen vor dem Import in dem Zielprojekt bereits definiert sein oder zuvor definiert werden.

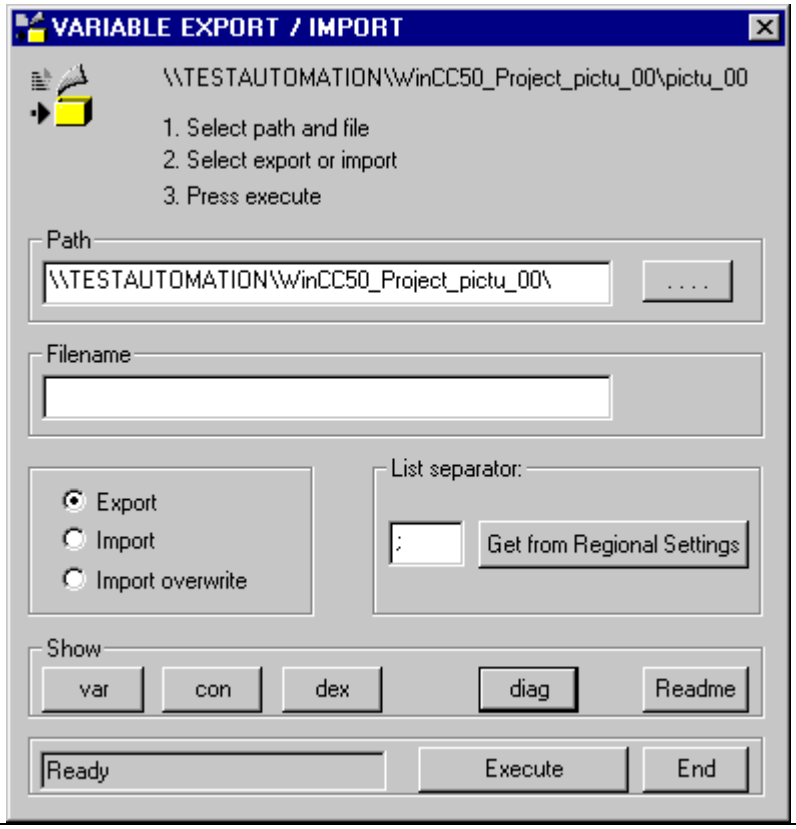
Hinweis:
Für eine automatische Erzeugung von Verbindungen sowie Dateneintragungen in der WinCC-Datenbank könnte ein spezielles Programm erstellt werden, das über die WinCC API-Schnittstelle derartige Definitionen automatisch vornimmt. Über diesen Weg können bereits bestehende Anlagendaten automatisch ergänzt werden. Dies muß von einem Fachmann für die WinCC API-Programmierung bzw. SQL-Programmierung erstellt werden.

Die im Variablenhaushalt definierten Variablen können jederzeit als Textdatei **exportiert** werden, um die Liste der Variablen zu ergänzen. Anschließend müssen diese erzeugten Daten wieder in den Variablenhaushalt des Projektes **importiert** werden. Die erzeugten Dateien sind im Format CSV (comma separated values) abgelegt und können von beliebigen Aufbereitungsprogrammen gelesen und weiterverarbeitet werden.
Dazu existiert eine eigens Anwendungsprogramm auf der WinCC CD-ROM unter dem Verzeichnis *\SmartTools\CC_VariablenImportExport*. Dieses Windows Programm wird als Zusatzprogramm bereitgestellt:

- zur Auslagerung der Daten des Variablenhaushalts
- für den Import von bereits extern erstellten Variablendaten
- für die Massendaten-Projektierung

Vorgehensweise

Für den Export bzw. Import der Daten müssen nun folgende Schritte durchlaufen werden:

Schritt	Vorgehen: Variablen importieren, exportieren
1	Öffnen Sie Ihr WinCC-Projekt im WinCC *Explorer*.
2	Definieren Sie nicht vorhandene, aber für den späteren Import nötige Verbindungen (Kanal-DLL – logische Verbindung – Verbindungsparameter). Dies sollte aber erst im neuen Projekt erfolgen. Hierzu ist evtl. ein zweiter Export-Import-Vorgang nötig.
3	Aktivieren Sie das Programm *Var_exim* über **D**. Die Benutzeroberfläche dieses Programms ist nachfolgend dargestellt. 

Import - Export

Für den Export bzw. Import sind folgende Einstellungen notwendig:

Ort, Aktion	Import	Export
Path	Das Projektverzeichnis mit den Dateien für den Variablenimport auswählen. Die Auswahl erfolgt durch Selektion der Datei <*projektname*>.*mcp*. Die Dateien mit den Daten für den Import müssen im selben Verzeichnis liegen wie die Projektdatei.	Das Projektverzeichnis für den Variablenexport angeben. Die Auswahl erfolgt durch Selektion der Datei <*projektname*>.*mcp*.
Aktion	*Import* anwählen. Sollen bereits vorhandene variablendaten überschrieben werden, ist *Import overwrite* anzuwählen.	*Export* anwählen.
Execute	Ausführen wählen Der anschließende Dialog zeigt die eingestellten Parameter und führt, nach Bestätigung mit *OK*, die Umsetzung durch.	Ausführen wählen Der anschließende Dialog zeigt die eingestellten Parameter und führt, nach Bestätigung mit *OK*, die Umsetzung durch.
Statusanzeige	End Export/Import	End Export/Import
Variablendatei *Name_vex.csv*	Basis für den Import: Besteht aus Kopfzeile und Datensätzen	Die erzeugte Variablenliste wird in Textform in dieser Datei abgelegt. Diese kann entweder direkt über die Taste *var* geöffnet werden oder mit einen Texteditor (Notepad) oder mit EXCEL bearbeitet werden.
Variablendatei *Name_cex.csv*	Basis für den Import: Besteht aus Kopfzeile und Datensätzen (Strukturkomponenten)	In dieser Datei stehen die aktuell konfigurierten Verbindungen, auf die in der Variablendatei Bezug genommen wird. Diese kann entweder direkt über die Taste *con* geöffnet werden oder mit einem Texteditor (Notepad) oder mit EXCEL bearbeitet werden.
Datenstruktur-datei *Name_dex.csv*	Basis für den Import: Besteht aus Kopfzeile und Datensätzen	Falls Variablen mit Datenstrukturtypen enthalten sind, entsteht zusätzlich diese Datei mit Strukturinformationen. Ihr Inhalt kann über einen Texteditor (Notepad) oder mit EXCEL bearbeitet werden.
Diagnosedatei *Diag.txt*	Diagnosedatei mit Hinweisen, welche Variablen nicht importiert werden konnten.	

Variablenliste

In der folgenden Tabelle ist der Aufbau der Variablenlisten beschrieben.

Feld	Typ	Beschreibung
Varname	Char	Variablenname
Conn	Char	Verbindung
Group	Char	Gruppenbezeichnung
Spec	Char	Interne Variable oder Adresse (passend zur Verbindungsart)
Flag	DWORD	
Common	DWORD	
Ctyp	DWORD	Variablentyp 1 BIT 2 SBYTE 3 BYTE 4 SWORD 5 WORD 6 SDWORD 7 DWORD 8 FLOAT 9 DOUBLE 10 TEXT_8 11 TEXT_16 12 Rohdatentyp 13 Feld 14 Struktur 15 BITFELD_8 16 BITFELD_16 17 BITFELD_32 18 Textreferenz
CLen	DWORD	Länge der Variablen
CPro	DWORD	Interne oder externe Variable
CFor	DWORD	Formatkonvertierung
Protocol		
P1	BOOL	Fehler in Obergrenze
P2	BOOL	Fehler in Untergrenze
P3	BOOL	Umwandlungsfehler
P4	BOOL	Schreibfehler
P5	BOOL	
P6	BOOL	
L1	BOOL	Ersatzwert bei Obergrenzenfehler
L2	BOOL	Ersatzwert bei Untergrenzenfehler
L3	BOOL	Startwert
L4	BOOL	Ersatzwert bei Verbindungsfehler
L5	BOOL	Obergrenze gültig
L6	BOOL	Untergrenze gültig

Feld	Typ	Beschreibung
L7	BOOL	Startwert gültig
L8	BOOL	Ersatzwert gültig
LF1	Double	Obergrenze
LF2	Double	Untergrenze
LF3	Double	Startwert
LF4	Double	Ersatzwert
Skalierung		
SCF	DWORD	1 wenn Skalierung definiert ist
SPU	Double	Wertebereich Prozeß von
SPO	Double	Wertebereich Prozeß bis
SVU	Double	Wertebereich Variable von
SVO	Double	Wertebereich Variable bis

Verbindungsliste

Feld	Typ	Beschreibung
Conname	Char	Logischer Verbindungsname
Unit	Char	Kanal Unit
Common	Char	Allgemein
Specific	Char	Spezifische Verbindungsparameter
Flag	DWORD	

Datenstrukturliste

Feld	Typ	Beschreibung
Datastructure	Short	Datenstrukturname oder Komponentenname
Type ID	Short	Identifikation (wird in Variablenliste unter Ctyp verwendet)
Creator ID	Short	

Um die Textlisten in EXCEL (Version 7.0 oder 8.0) weiterverarbeiten zu können, müssen Sie die exportierten Dateien mit dem Dateityp *Textdateien [*.prn; *.txt; *.csv]* öffnen.

Vorschriften

Die Anpassung der Variablendaten in der Textliste erfolgt unter folgenden speziellen Vorschriften:

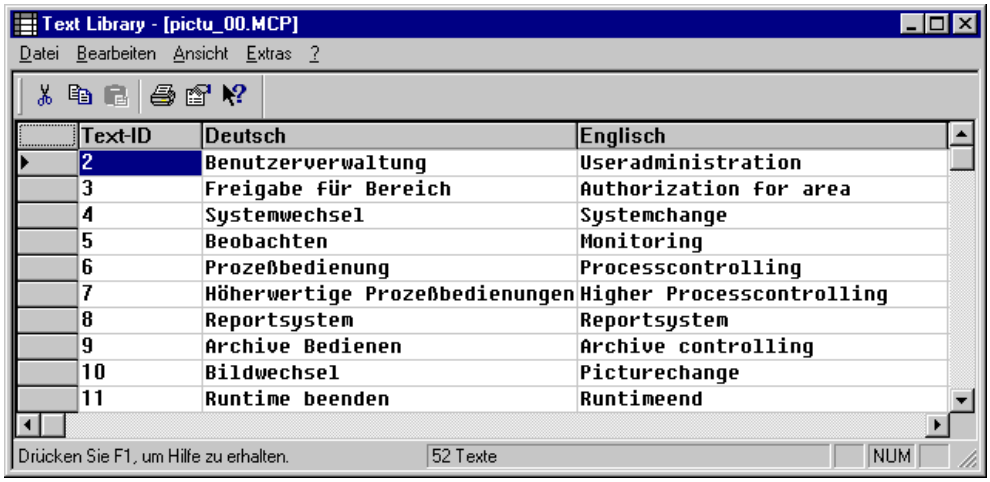
Typ	in Textliste	in WinCC
Verbindungen	allgemeine Beschreibung	Logische Verbindungen **müssen**, falls nicht vorhanden, neu definiert werden!
	Spezielle Beschreibung Kanal-DLL	Kanal-DLL **müssen**, falls nicht vorhanden, neu definiert werden!
Variablen-gruppe	Keine Gruppeninformation Sind im Projekt Gruppen definiert, die keine Variable enthalten, so werden diese Gruppen auch nicht exportiert.	Die Gruppeninformation wird beim Import mit der ersten Variablen einer Gruppe automatisch generiert.
Variablen	Allgemeine Beschreibung	
	Spezifische Beschreibung jeweilige Kanal-DLL oder interne Variable	Kanal-DLL oder interne Variable
	Beim Export werden die fehlenden Anteile durch * ersetzt.	Variablen mit fehlender spezifischer Beschreibung werden nicht importiert!
Variablen vom Typ Daten-struktur	Zuordnung entsprechend der Datenstrukturdefinition der Strukturliste	Wird dem Datentyp zugeordnet.
Datenstruktur Definition		Muß vom Projekteur definiert werden.
Grenzen	Werden nicht über die Textliste ex- oder importiert	Muß vom Projekteur definiert werden.

Bevor Sie den Import der geänderten bzw. neuen Variablen vornehmen, führen Sie zuerst eine **Datensicherung** Ihres Projektes durch. Es werden Veränderungen in der Datenbank vorgenommen. Diese Datenbankveränderungen können in WinCC nicht rückgängig gemacht werden.

3.3.9.6 Übernahme von mehrsprachigen Texten (aus Bildern, in Meldungen)

Mehrsprachige Texte der Bilder

Die in den Bildern hinterlegten mehrsprachigen Texte können durch das Kopieren des Bildes selbst in das neue Projekt übernommen werden. In dem neuen Projekt muß die jeweilige Sprache in der Textbibliothek angefügt werden. Prüfen Sie dazu die Einstellungen in der Text Library. Für jede Sprache muß eine Spalte vorliegen!



Text-ID	Deutsch	Englisch
2	Benutzerverwaltung	Useradministration
3	Freigabe für Bereich	Authorization for area
4	Systemwechsel	Systemchange
5	Beobachten	Monitoring
6	Prozeßbedienung	Processcontrolling
7	Höherwertige Prozeßbedienungen	Higher Processcontrolling
8	Reportsystem	Reportsystem
9	Archive Bedienen	Archive controlling
10	Bildwechsel	Picturechange
11	Runtime beenden	Runtimeend

Sollen in dem neuen Projekt die Sprachen nur teilweise übernommen werden?
Da die gesamte Textinformation pro Bild hinterlegt wird, muß pro Bild eine Nachprojektierung in der jeweiligen Sprache vorgenommen werden. Es ist dabei zu überlegen, ob es sinnvoll ist, die bereits projektierten Texte wieder zu entfernen. Das Umschalten zur Laufzeit wird nur über speziell projektierte Tasten ermöglicht und könnte somit für das Projekt eingeschränkt werden. Sollen die Texte für eine bereits eingeführte Sprache trotzdem entfernt werden, so wird der Export und Import über das Zusatz-Tool *language* empfohlen.

Bilder mit Textreferenzen übernehmen

Werden in den übernommenen Bildern Textreferenzen genutzt, müssen folgende Daten mit übernommen werden:

- Übernahme der zugehörigen Variablen (Export oder Neudefinition) aus dem Variablenhaushalt des WinCC-Projektes
- Übernahme der Texte aus der Textbibliothek
- Die Textreferenzvariablen müssen mit den gültigen Text-Identifikationsnummern (Text-ID's) versehen sein. Prüfen Sie, ob die Text-ID's noch mit den zugehörigen Texten übereinstimmen.

Übernahme von Texten aus der Textbibliothek

Werden die Texte aus der Textbibliothek nur teilweise übernommen, so muß eine Anpassung der Text-IDs vorgenommen werden. Die Übernahme von Texten aus der Textbibliothek ist über den Export/Import-Mechanismus im Editor *Text Library* möglich.

3.3.9.7 Übernahme von Meldungen

Die Informationsgrundlage für Meldungen (Alarme) ist

- änderungsbedürftig und
- aufgrund der Menge (Massendaten) projektierungsaufwendig.

Aus diesem Grunde wird die Übernahme von Meldedaten aus bisherigen Projekten sehr häufig genutzt. Abhängig von der Quelle der Meldedaten sind folgende Übernahmewege möglich:

- Übernahme der bereits projektierten Meldeinformationen aus Vorgänger-Systemen (z.B. COROS)
- Import von (Einzel-)Meldungen aus einem existierenden WinCC-Projekt
- Import von Meldeinformationen aus der Konzeptphase

Übernahme von Meldungen aus COROS

Die Vorgehensweise für die aufgeführten Quellen sieht folgendermaßen aus:

Schritt	Vorgehen: Übernahme vorhandener COROS Meldungstexte
1	In COROS Meldeinformationen exportieren (*mldtexte.txt*)
2	In WinCC über den *Dynamic Wizard Importfunktionen* → *Import Meldungen* diese Meldedatei einlesen. Die Daten werden nun in das aktuelle WinCC-Projekt importiert. 

Übernahme von Meldungen aus einem WinCC-Projekt

Werden Meldungen aus einem vorhandenen Projekt übernommen, so muß zunächst geklärt werden, ob das Ziel-Meldesystem in der Datenbankstruktur in der selben Form aufgebaut wurde oder nicht. Die Unterschiede sind z.B. in den Anwender- und Prozeßwertblöcken zu sehen. Organisieren Sie die Ziel-Datenblöcke in der Reihenfolge (als auch in der Länge der Textelemente) möglichst gleich. Ansonsten müssen Sie vor dem Import eine Anpassung in den einzelnen Spalten vornehmen.

Schritt	Vorgehen: Übernahme vorhandener WinCC Meldungstexte
1	Öffnen Sie den Editor *Alarm Logging* im aktuellen Projekt.
2	Rufen Sie den Export der Meldungen über den Menüpunkt *Meldungen → Einzelmeldungen exportieren* auf.
3	Nennen Sie die Ziel-Textdatei, in welche die zu exportierenden Meldeinformationen abgelegt werden sollen. Selektieren Sie über *Selektion* die zu exportierenden Meldungen über die Kriterien z.B. Meldenummer, Meldeklasse.
4	Starten Sie den Exportvorgang über *Export*. Es entsteht nun eine Textdatei mit Meldeinformationen, die jeweils mit Komma getrennt sind.
5	Schließen Sie das aktuelle Projekt und öffnen Sie das neue Projekt. Öffnen Sie wieder den Editor *Alarm Logging* und definieren Sie zunächst die nötigen Meldeklassen und Meldearten. Pro Meldeart definieren Sie eine Meldung, um das Rohgerüst für den folgenden Import zu erhalten.
6	Für den Export dieser Basismeldungen rufen Sie *Meldungen → Einzelmeldungen exportieren* auf. Gehen Sie entsprechend Schritt 3 und 4 vor.
7	Öffnen Sie nun z.B. in EXCEL die Meldedatei des Quellprojektes und die Meldedatei des Zielprojektes. Die Spalten sind jeweils mit Komma getrennt. Vergleichen Sie den Aufbau der Meldeblöcke und nehmen Sie eventuell Anpassungen durch umsezten oder umbenennen von Spalten vor. Tragen Sie in den Blöcken mit den Text-IDs jeweils den Index 0 ein. Über diesen Weg werden die Texte durch den Import automatisch in der Textbibliothek organisiert. Die alten Ident-Nummern dürfen auf keinen Fall beibehalten werden! Die geänderte Datei muß erneut als Textdatei gespeichert werden
8	Rufen Sie nun den Importvorgang über *Meldungen → Einzelmeldungen importieren* auf.
9	Nennen Sie nun die Quell-Textdatei mit den exportierten Meldeinformationen. Im folgenden müssen Sie entscheiden, ob beim Import bereits vorhandene Meldungen überschrieben werden sollen. Die Zuordnung erfolgt über die Meldenummer, die im Projekt eindeutig definiert sein muß.
10	Die Meldungen werden anschließend importiert und ergänzen Ihr vorhandenes Meldesystem mit den bereits projektierten Meldeinformationen. Prüfen Sie die importierten Zuordnungen.

Hinweis:
Werden Meldedaten aus einem WinCC-Projekt der V. 1.10 übernommen, müssen die Spaltenüberschriften in der Meldetextdatei beachtet werden!

Import von Meldeinformationen aus der Konzeptphase mittels EXCEL-Tabellen

Die Meldeinformationen liegen in einer EXCEL-Tabelle bereits vor. Durch Übernahme der Spalten in die Meldestruktur des WinCC-Projektes können diese Meldungen übernommen werden. Dazu muß auf einer WinCC Meldedatei aufgebaut werden. Diese wird folgendermaßen erzeugt:

Schritt	Vorgehen: Meldestruktur anlegen
1	Öffnen Sie das neue Projekt im WinCC *Explorer*. Öffnen Sie den Editor *Alarm Logging* und definieren Sie zunächst die nötigen Meldeblöcke, Meldeklassen und Meldearten. Pro Meldeart definieren Sie eine Meldung, um das Rohgerüst für den folgenden Import zu erhalten.
2	Für den Export dieser Basismeldungen rufen Sie *Meldungen → Einzelmeldungen exportieren* auf.
3	Nennen Sie die Ziel-Textdatei, in welche die zu exportierenden Meldeinformationen abgelegt werden sollen.
4	Starten Sie den Exportvorgang über *Export*. Es entsteht nun eine Textdatei mit den Meldeinformationen, die durch Komma getrennt sind.
5	Öffnen Sie in EXCEL die Meldedatei und die neu erzeugte Meldedatei des Ziel-Projektes. Die Spalten sind durch Komma getrennt. Legen Sie jeweils eine kopierte Zeile zu der entsprechenden Meldeklasse/Meldeart in der Tabelle an. Übernehmen Sie aus den Quelldaten die Meldetexte, etc. und tragen Sie diese in den zugehörigen Blöcken ein. Zum Beispiel: *Block 1 → Meldetext*. **Nummerieren** Sie alle Meldezeilen (z.B. von 1 ab) durch. Dies kann in EXCEL sehr schnell mit Hilfe der Aufzählungen in der Meldenummernspalte erzeugt werden.
6	Tragen Sie in den Blöcken mit den Text-ID´s jeweils den Index 0 ein. Über diesen Weg werden die Texte durch den Import automatisch in der Textbibliothek organisiert. Die alten Ident-Nummern dürfen auf keinen Fall beibehalten werden!! Die geänderte Datei muß wiederum als Textdatei gespeichert werden.
7	Rufen Sie im Alarm Logging Editor den Importvorgang über *Meldungen → Einzelmeldungen importieren* auf.
8	Nennen Sie die Quell-Textdatei mit den exportierten Meldeinformationen. Im folgenden definieren Sie die Parameter so, daß bereits vorhandene Meldungen überschrieben werden. Die Zuordnung erfolgt über die Meldenummer, die im Projekt eindeutig sein muß.
9	Die Meldungen werden nun importiert und ergänzen Ihr vorhandenes Meldesystem mit den bereits projektierten Meldeinformationen. Prüfen Sie die importierten Zuordnungen.

3.3.9.8 Übernahme von Meßwerten

Da die Festlegung von Meßpunkten sowie die Definition der Prozeßwertarchive als auch der Anwenderarchive mit Ihren Eigenschaften direkt in die Datenbankstruktur integriert sind, ist eine Übernahme (ohne direkte Datenbankzugriffe mittels fundierter Datenbankkenntnisse) nicht möglich. Das heißt, daß eine Neuprojektierung dieser Archive und Meßpunkte erfolgen muß oder zu Beginn der Projektierung die Daten aufgrund der Kopie eines gesamten Basis-Projektes automatisch übernommen werden.

3.3.9.9 Übernahme von Drucklayouts

Kopieren Sie die gewünschten Drucklayouts, **.rpl* bei Seitenlayouts oder **.rp1* bei einem Zeilenlayout, aus dem Quellverzeichnis in das Verzeichnis *\PRT* des neuen Projektes.

3.3.9.10 Übernahme von globalen Aktionen

Kopieren Sie die gewünschten globalen Aktionen oder Hintergrundaktionen **.pas* aus dem Quellverzeichnis in das Verzeichnis *\Pas* des neuen Projektes.

3.3.9.11 Übernahme von Projektfunktionen

Kopieren Sie die gewünschten Projektfunktionen **.fct* aus dem Quellverzeichnis in das Verzeichnis *\Library* des neuen Projektes. Um diese Funktionen im Projekt bekanntzugeben, muß im Editor *Global Scripts* der Menüpunkt *Optionen → Header generieren* aktiviert werden. Eine detaillierte Beschreibung hierzu finden Sie im Kapitel 4.1 Entwicklungsumgebung für C Scripte.

3.3.9.12 Nutzen von Standard-Funktionen

Standard-Funktionen müssen im Gegensatz zu *Projekt-Funktionen* nicht zusätzlich kopiert werden. Diese Funktionen sind sofort für das Projekt verfügbar, da diese Funktionen für alle WinCC-Projekte auf der Station bekannt sind.

3.3.9.13 Übernahme der Benutzerverwaltung

Da die Festlegungen für Benutzergruppen, Benutzer und Zugriffsrechte direkt in die Datenbankstruktur integriert sind, ist eine Übernahme nicht möglich. Aus diesem Grunde ist eine Neuprojektierung erforderlich.
Eine Ausnahme bietet zu Projektierungsbeginn eine automatische Datenübernahme aufgrund der Kopie eines gesamten Basisprojektes.

3.3.10 Mauslose Bedienung

Die Bedienung der Anlagenbilder erfolgt unter WinCC in vielen Fällen über die Maus. Der Mausklick ist das Ereignis unter den jeweiligen Dynamisierungen, der am häufigsten und in den unterschiedlichsten Varianten (Mausklick links bzw. rechts drücken oder loslassen) eingesetzt wird. Es existieren aber auch Anlagen, die in gemischter Form oder nur mit Tastatur bedient werden. Operator Panels zum Beispiel werden nur über Tastatur bedient.

3.3.10.1 Bedienung über Tastatur

Die Bedienung über Tastatur unterscheidet dabei folgende Eingabemöglichkeiten:

- Funktionstasten F1 bis F12
- Spezielle Funktionstasten (z.B. Operator Panel Funktionstasten SF10)
- Standard Tastatureingaben
- Bewegungen über Eingabefelder bzw. Bedientasten mittels Richtungstasten bzw. über spezielle Tasten

Die Projektierung der Bedienungen ohne Maus muß für folgende Projektierungsbereiche im einzelnen betrachtet werden:

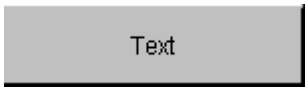
- Bedientasten im Anlagenbild (z.B. für Bildwechsel)
  - über Funktionstasten
  - über spezielle Tasten
  - über Standard Tasten
- beliebige Tastenbedienung
- Bewegung über Bedienobjekte
- Eingabefelder im Anlagenbild
  - Ein-/Ausgabefelder
  - spezielle Eingabeobjekte (Checkbox, ...)

- Alarm Logging Controls (Meldefenster)
  - Bedienungen über Funktionstasten
  - Bedienungen über eigens projektierte Tasten

- Tag Logging Controls (Kurven- oder Tabellenfenster)
  - Bedienungen über Funktionstasten
  - Bedienungen über eigens projektierte Tasten
- Druckauftrag über Taste anstoßen
- an- oder abmelden über Tastatur

Bedientasten werden im typischen *Windows-Stil* projektiert. Aus diesem Grunde finden Sie die Standard Windows-Bedientaste in der Objektpalette vor. Diese Taste kann jederzeit über weitere grafische Elemente ergänzt werden.

Bedientasten

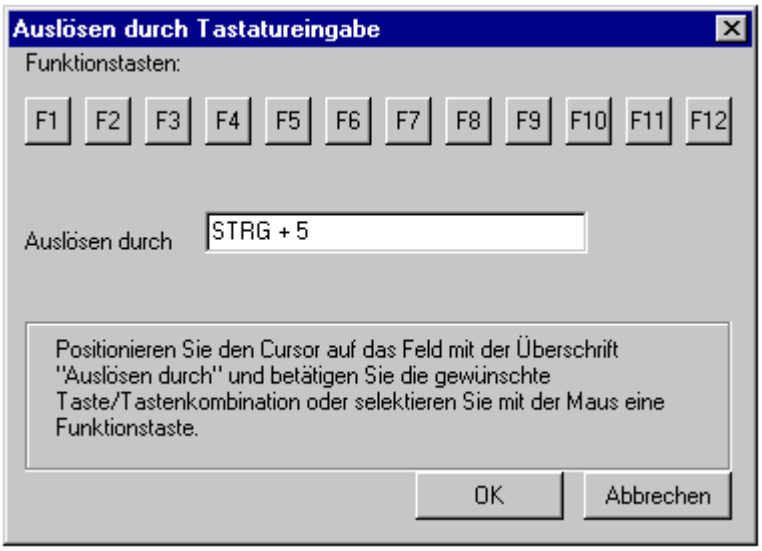


Bedienung über Funktionstasten

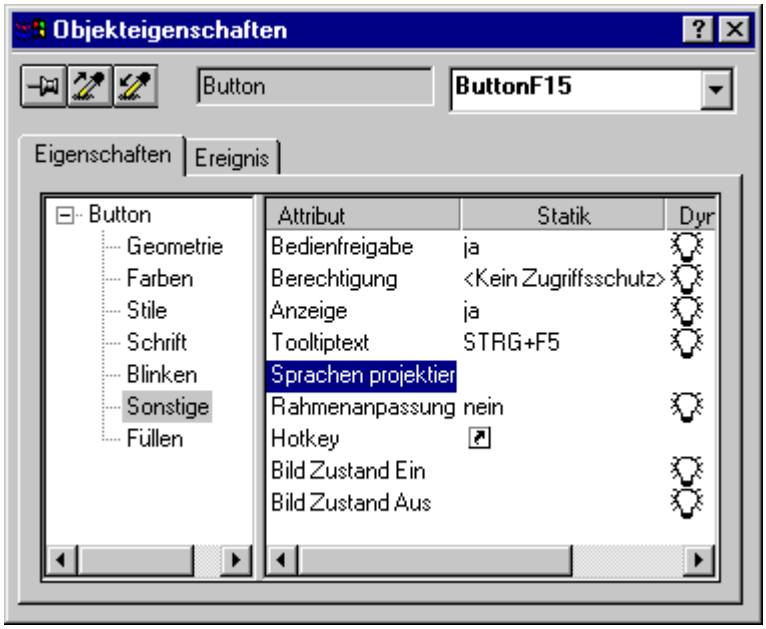
Die Funktionstasten F1 bis F12 auf der Standard-Tastatur werden häufig als (zusätzliche) Tastaturbedienung für Bedientasten zum Bildwechsel in der Anlagenbildhierarchie genutzt. Diese Funktionstasten können zu den projektierten Windows-Buttons als sogenannte Hotkeys jederzeit projektiert werden. Ein Hotkey legt die Schnellbedienung für die jeweilige Funktion fest.

Als Hotkeys können z.B. die genannten Funktionstasten hinterlegt werden. Diese werden bereits als Auswahlknöpfe in der Projektierungsmaske angeboten.

Wird eine Kombination z.B. mit der UMSCHALT-Taste oder der STRG-Taste benötigt, so muß in dem Einstellungsfeld die gewünschte Tasten-Folge (z.B. UMSCHALT, F2) direkt durch Drücken der jeweiligen Tasten eingegeben werden. Es müssen keine Spezial-Codes eingegeben werden.
Diese gewählte Kombination wird in dem Eingabefeld aufgezeigt.



Diese Tastenwahl wird in den Eigenschaften des Objektes hinterlegt und kann entweder über den Konfigurationsdialog oder über die Eigenschaft *Sonstige → Hotkey* direkt modifiziert werden.



Aktion für Hotkey

Die Aktion, die aufgrund der Funktionstasteneingabe (Hotkey-Tastenfestlegung) ausgelöst werden soll, muß unter den Ereignissen des Windows-Buttons hinterlegt werden. Das Ereignis hierfür muß der **Mausklick** sein. Das Ereignis wird beim Loslassen der Maustaste ausgelöst, aber nur dann, wenn sich der Mauszeiger beim Drücken und Loslassen der Maustaste über dem Objekt befindet. Liegt hinter dem Ereignis *Mausklick* keine Aktion, sondern z.B. nur hinter dem (ähnlichen) Ereignis *linke Maus drücken*, wird die Aktion über die Funktionstaste **nicht** ausgelöst! Beachten Sie auch bei der Projektierung, daß die Funktionstaste im Bild nur einmal verwendet werden kann.

Spezielle Funktionstasten

Werden spezielle Funktionstasten, z.B. die Tasten des Operator Panels F13, S1 etc. für die Bildbedienung eingesetzt, so müssen diese Tasten auf Tastenkombinationen umgelegt werden. Die Taste F13 könnte z.B. auf die Kombination UMSCHALT F1 gelegt werden. Die gewählten Tastenkombinationen werden neben der oben beschriebenen Verwendung in der Visualisierung zusätzlich gerätespezifisch definiert. Hierzu finden Sie spezielle Tastatur-Einstellungen abhängig von den jeweilig eingesetzten Geräten vor. Zum Beispiel wird für den Industrie-PC eine Datei *FI25.key* für die Einstellungen der Tastencodes zur Verfügung gestellt. In diesen gerätespezifischen Dateien werden die Codes für die Funktionstasten hinterlegt. Nach der Anpassung der Geräte-Tastaturdefinition - pro Funktionstaste der jeweilige hexadezimale Code - sowie Aktivierung der neuen Tastaturcodes können diese Tasten in den Anlagenbildern für die Bildbedienung eingesetzt werden.

Standardtasten

Wird die Aktion nicht auf eine Funktionstasteneingabe bezogen, sondern auf eine ausgewählte Taste der Standard-Tastatur z.B. den Buchstaben *m*, so wird diese Taste als Hotkey im Objekt *Windows-Button* hinterlegt.

Beliebige Tastenbedienung

Das Design der Bedientasten kann auch selbst für die Anlagenbilder entworfen werden. Weitere Bedientasten finden Sie z.B. in der Anwenderbibliothek unter *3D-Tasten* oder *Anwenderobjekte*.
Eigens entworfene Objekte, die nicht auf dem Windows-Button basieren, können nicht über den Hotkey projektiert werden. Alle anderen Objekte müssen über das *Tastenereignis* des Objektes für die Tastenbedienung projektiert werden. Hierzu gibt es pro Objekt folgende Tastatur-Ereignisse:

- drücken
- loslassen

Dieses Tastenereignis muß für die mögliche Projektierung der Tastaturbedienung verfügbar sein. Beim Einsatz von vordefinierten Tasten aus der Anwenderbibliothek ist daher zunächst zu prüfen, ob diese Taste sich für die Bedienung ohne Maus eignet. Die Umschalt-Buttons der Anwenderobjekte z.B. sind nicht immer für die Tastaturbedienung freigegeben. Eine Anpassung kann daher nötig sein.

Vorprojektierte Tastenbedienungen (z.B. Blättern in der Bildhierarchie) finden Sie in den Optionspaketen (z.B. *Basic Process Control - Picture Tree Manager* etc.).

Wird eines dieser Objekte als Bedienelement genutzt, so wird die auslösende Taste hinter dem Ereignis *Tastatur - drücken* oder *Tastatur - loslassen* projektiert. Als Aktion kann entweder eine *Direktverbindung* oder eine *C-Aktion* projektiert werden.
Das auslösende Tastenereignis ist entweder eine

- beliebige Tastenaktion oder eine
- ausgewählte Taste der Standard-Tastatur.

Handelt es sich um eine beliebige Taste, so kann die *Direktverbindung* eingesetzt werden; muß hingegen eine spezielle Tasteneingabe geprüft werden, so muß eine *C-Aktion* eingesetzt werden. Die *C-Aktion* prüft den eingegebenen Tastencode vor der Fortführung der eigentlichen Aktionsfolge zeichenweise.

3.3.10.2 Bewegung über Bedienobjekte (Eingabefelder und Bedienfelder)

Mit der Maus kann jedes bedienbare Objekt direkt angeklickt werden. Die Bedienbarkeit wird jeweils durch den veränderten Mauszeiger visualisiert. Wie können diese Objekte ohne Maus bedient werden?

Alpha- Schaltcursor

Die bedienbaren Objekte können im Runtime Modus über die Bewegungstasten angesprungen werden. Man unterscheidet zwischen:

- Alphacursorobjekten (EA-Objekte) und
- Schaltcursorobjekten.

Ein-/Ausgabe-Objekte werden über den Alphacursor (Tabulator-Taste bzw. UMSCHALT Tabulator-Taste) angewählt.
Alle Bedienelemente (mit Maus-, Tastatur- oder beliebiger Bedienung) können in die Bedienung über den Schaltcursor einbezogen werden. Die Ein-/ Ausgabefelder können sowohl in die Alphacursor- als auch die Schaltcursor-Bedienung integriert werden.

TAB-Reihenfolge

Durch die sogenannte *TAB-Reihenfolge* (über Menüpunkt *Bearbeiten →TAB-Reihenfolge →Alphacursor oder →Schaltcursor* einstellbar) haben Sie Einfluß darauf, in welcher Reihenfolge bedienbare Objekte im Runtime Modus angesprungen werden. Das aktuell angewählte Objekt kann im Runtime visualisiert werden. Dies ist der Runtime-Cursor, der auch abgeschaltet werden kann (*Rechnereigenschaften → Graphics Runtime).* Buttons, welche im Windows-Stil dargestellt werden, werden bei Anwahl immer mit einem gestrichelten Viereck in der Taste gezeigt.

Das Bewegen über die bedienbaren Elemente ist abhängig von den Einstellungen für das Graphics Runtime (*Rechnereigenschaften → Graphics Runtime).*

Bewegung	**Standard-Tasten**	**Tasteneinstellungen**
Auf, Ab links, rechts	Pfeiltasten oder Tabulator (nächstes) oder UMSCHALT Tabulator (vorheriges)	weitere Tasteneinstellungen über *Rechnereigenschaften → Graphics Runtime →Cursorsteuerung-Tasten*
Alpha-/Schalt-cursor	Schaltcursor	Umschalten zwischen Alpha- und Schaltcursor über Hotkeys (*Rechnereigen-schaften → Graphics Runtime →Hotkeys)* oder eigene Tasten (mit Verwendung der internen Funktion *SetCursorMode*)
Tabellen-bewegung (Cursorgruppe)	Normal, d.h. jeweils zeilenweise Bearbeitung wenn der Cursor das Ende der Cursorgruppe erreicht hat, bleibt er an dieser Position stehen.	Änderung des Verhaltens über *Rechnereigenschaften → Graphics Runtime →Cursorsteuerung-Verhalten*

Ein-/Ausgabefelder

Projektierte Ein-/Ausgabefelder sind nach der Auswahl über die Tastatur direkt bedienbar, d.h. Sie können sofort Ihre neuen Daten eingeben. Ein reines Ausgabefeld (*Eigenschaften → Ausgabe/Eingabe → Feldtyp→Ausgabe)* kann nicht bedient werden.
Das Bestätigen der Eingabe erfolgt abhängig von der projektierten Eigenschaft (*Eigenschaften → Ausgabe/Eingabe →Übernahme bei Verlassen*) über die EINGABE-Taste. Die Abbruchtaste (ESC) dagegen beendet die Eingabe, ohne daß eine Änderung des Wertes vorgenommen wird.

Weitere Eingabeobjekte

Neben den typischen Analog-Eingabefeldern sind unter den Windows-Anwendungen noch weitere Eingabemöglichkeiten bekannt. Diese speziellen Objekte finden Sie in der Objektpalette unter *Windows-Objekte*

- Checkbox
- Radiobox

Das Setzen der einzelnen Auswahlfelder der Check- oder Radiobox erfolgt über die Leertaste und das Bewegen über die einzelnen Komponenten in der Box mittels der Auf-/Ab-Tasten (z.B. Richtungstasten). Dies ist die bereits standardmäßig hinterlegte Tastenbelegung.
Ein weiteres Eingabeobjekt stellt das **Textlistenobjekt** dar. Über eine, abhängig von den projektierten Einträgen, aufgeschlagene Liste kann eine Auswahl getroffen werden:

Die Bedienung kann ebenfalls über die Standard-Tastatur erfolgen. Es muß keine spezielle Projektierung für die Tastaturbedienung hinterlegt werden.
Das Aufschlagen der Liste erfolgt über die EINGABE-Taste, das Bewegen in der Liste über die Auf-/Ab-Tasten und das Bestätigen der aktuellen Auswahl über die EINGABE-Taste.

Weitere Eingabe-Objekte könnten über die OCX-Elemente in WinCC genutzt werden. Deren Bedienung und Projektierung ist jedoch abhängig von den verfügbaren Ereignissen und Eigenschaften, die objektbezogen definiert sind. Dies muß im Einzelfall geklärt werden.

3.3.10.3 Alarm Logging Funktionstasten zu den Toolbar-Tasten

In den Alarm Logging Controls (Meldefenstern) werden unterschiedliche Bedientasten in der Toolbar eingestellt, die standardmäßig über die Maus bedient werden.

Die häufigsten Bedienungen in einem Meldefenster sind

- Anwahl einer Meldung für die Quittierung
- Auf-/Abwärts-Bewegen in der Meldeliste
- Blättern in der Meldeliste

Hinweis:
Beim Aufschlagen des Meldefensters bzw. durch eine weitere Bedienung muß die Bedienbarkeit im Meldefenster und nicht im Hauptfenster liegen. Abhängig von der aktuellen Bedienbarkeit wirken die Tastenbedienungen (bzw. Funktionstasten) auf die Funktionstastenleiste des Hauptfensters oder auf die hinterlegten Tastenbedienungen des Meldefensters.

Dies kann zum Beispiel über das Setzen des aktuellen Bedienfokus in diesen Fensterbereich erreicht werden. Der Bedienfokus wird normalerweise über die Maus durch das Anklicken gesetzt.
Über die Tastatur kann der Fokus über folgende projektierbaren Wege in Meldefenster gesetzt werden:

- Wechseln des Fensters über Hotkey
- Setzen des Fokus über eine Bedientaste oder
- direktes Setzen des Fokus auf ein definiertes Element im Meldefenster bei Bildanwahl.

Das Wechseln in das Meldefenster mittels einer Schnellbedienung (Hotkey), die für alle Fensterwechsel, d.h. auch bei Kurvenfenstern z.B. in gleicher Art eingesetzt werden kann, wird in den Anlauf-Parametern des Graphics Runtime festgelegt. Unter *Rechnereigenschaften→Graphics Runtime →Hotkeys →Fenster umschalten* wird die Tastenkombination (z.B. STRG W) eingetragen.

Nach dem Anwählen des Meldefensters kann über diese Tastenfolge direkt die Bedienung der Tasten der Toolbar erfolgen.

Das direkte Setzen des Bedienfokus auf das Meldefenster dagegen wird mittels der Internen Funktion *Set_Focus* realisiert. Eine *C-Aktion* zu einer Tastenbedienung oder bei Bildanwahl (*Bild-Objekt→ Ereignis→Sonstige →Bildanwahl*) kann daher das Aktivieren der Bedienung des Meldefensters beeinflussen.

Die Funktionen für den Bildfokus finden Sie unter *Interne Funktionen →graphics →Set →focus*. Zum Beispiel wird für das Setzen des Bedienfokus folgende Funktion aufgerufen:

```
Set_Focus(lpszPictureName,lpszObjectName);
```

Für die Parameter muß der Name des Hauptfensters (Bildname) sowie der des Alarm Controls (Objektname) eingetragen werden.

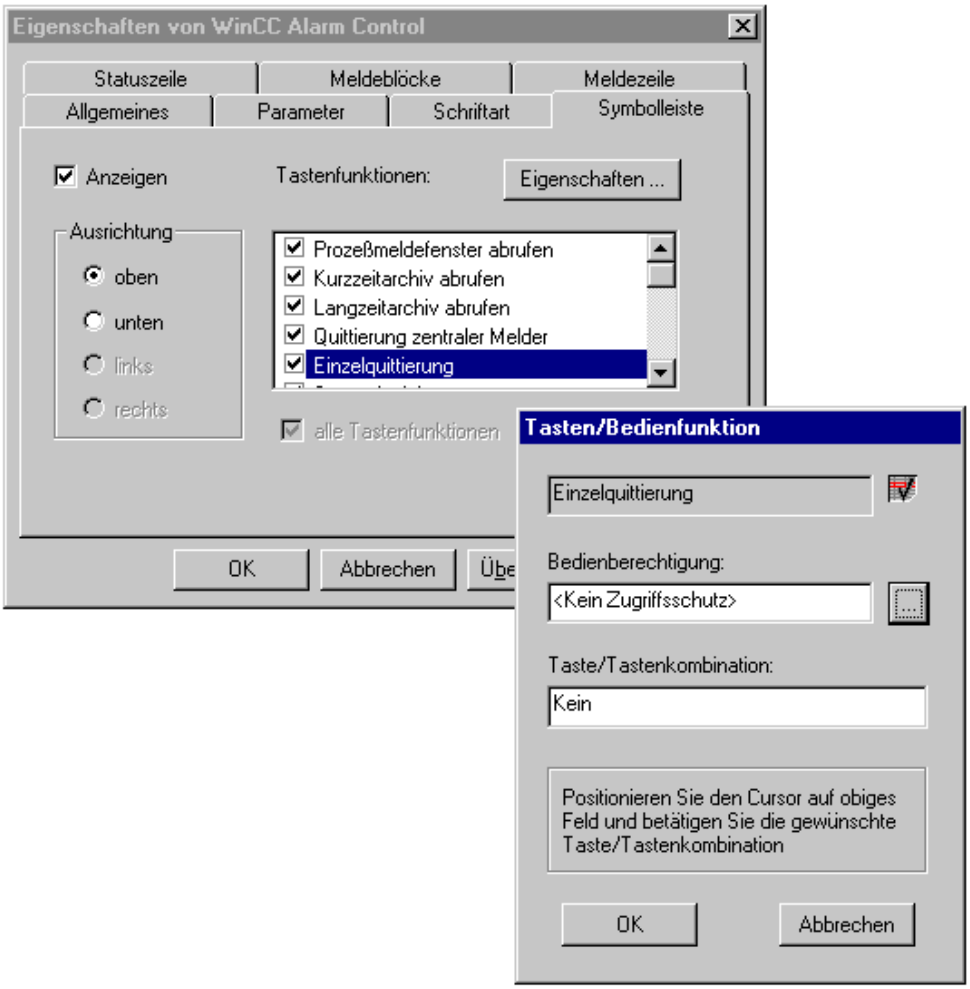
Die Anwahl einer Meldung im Meldefenster erfolgt über die Anwahl der Meldezeile. Bei der Anwahl des Meldefensters steht der aktuelle Cursor auf der jüngsten Meldung (letzte Meldung im Meldebild). Ob in dem Fenster eine Meldung angewählt werden kann bzw. ob das Blättern in den Meldungen möglich ist, hängt von der Aktivschaltung des Scroll-Mechanismus ab.

Das Ein-/Ausschalten des Bild-Scrollens kann über eine eigene Taste in der Toolbar umgeschaltet werden oder direkt in der Projektierung des *WinCC Alarm Control* eingestellt werden. Das aktive Einschalten des Scroll-Mechanismus bei Bildaufschlag erfolgt im Editor *Graphics Designer* → **D** *WinCC Alarm Control* → *Parameter* → *AutoScrolling*.

Ist das Scrollen sowie der Bedienfokus in dem Meldefenster aktiv, können die Bewegungen folgendermaßen ausgeführt werden:

Bewegung	Standard-Tasten	Tasteneinstellungen
Auf, Ab in der Meldeleiste	Richtungstasten	einzelne Meldezeilen
Anfang, Ende in der Meldeliste	Pos1-, Ende-Taste	Anfang bzw. Ende des Meldebildes
Blättern	Blättertasten	mehrere Meldezeilen

Um die Bedientasten der Toolbar wie z.B. die Quittierung der angewählten Meldung zu aktivieren, kann neben der Standard-Mausbedienung auch eine Funktionstastenbedienung definiert werden.
Für jede in der Toolbar eingeblendete Taste kann eine zugehörige Tastenbedienung in den Eigenschaften hinterlegt werden.



Über diese Projektierungsschritte können alle eingesetzten Bedientasten für das Meldefenster mit einer Tastenkombination versehen werden. Eine Tastenbedienung im Meldefenster ist daher zusätzlich zu definieren.

Alarm Logging anlagenspezifisch entworfene Toolbar-Tasten

Die gesamten Toolbar-Tasten sind von WinCC vorgegeben und können nicht verändert werden. Wird für die zu projektierende Anlage ein vorgegebenes Layout der Tasten vorgeschrieben, so muß die WinCC Toolbar deaktiviert werden (d.h. keine Toolbar) und die zugehörigen Tasten selbst entworfen werden. Alle diese neuen Tastenobjekte können nach den Wunschvorstellungen des Kunden z.B. mit Bildern versehen werden.

Die Funktionalität zu den einzelnen Tasten muß aber noch als zugehörige Aktion projektiert werden. In der *C-Aktion* des zugehörigen Ereignisses (z.B. Taste drücken) muß die entsprechende *Standard-Funktion* aus dem Funktionsbaum ausgewählt werden.

Die für die Tastenbedienung zur Verfügung gestellten Funktionen finden Sie unter *Standard-Funktionen → alarm.* Für jede Taste aus der Toolbar findet sich eine korrespondierende Funktion in der Liste wieder. Zum Beispiel wird für die Quittiertaste folgende Funktion aufgerufen:

```
AXC_OnBtnSinglAckn(lpszPictureName,lpszObjectName);
```

Für den Parameter muß der Fenstertitel des Alarm Logging Controls eingetragen werden. Diese Aktionen können auch für selbstentworfene Bedientasten über Mausbedienung genutzt werden.
Einige derartige Tastenbeispiele finden Sie in den Optionspaketen zum Alarmsystem (z.B. Basic Process Control - Hupenquittierung etc.).

3.3.10.4 Tag Logging Funktionstasten zu den Toolbar-Tasten

In den Kurven- bzw. Tabellenfenstern zur Darstellung von Meßwerten - auch Trendanzeigen genannt - werden unterschiedliche Bedientasten in der Toolbar eingestellt, die standardmäßig über die Maus bedient werden.

Die häufigsten Bedienungen in einem Kurvenfenster sind

- Blättern in den Meßwerten (Zeitachse)
- Zeitbereich auswählen
- Kurven auswählen
- Leselineal bedienen

Nach dem Aufschlagen des Kurvenfensters wird abhängig von der Projektierung der aktuelle Kurvenverlauf angezeigt.

Hinweis:
Beim Aufschlagen des Kurven- oder Tabellenfensters muß die Bedienbarkeit in diesem und nicht im Hauptfenster liegen. Abhängig von der aktuellen Bedienbarkeit wirken die Tastenbedienungen (bzw. Funktionstasten) auf die *Funktionstastenleiste* des Hauptfensters oder auf die hinterlegten Tastenbedienungen des Kurven- bzw. Tabellenfensters.

Dies kann zum Beispiel über das Setzen des aktuellen Bedienfokus in diesen Fensterbereich erreicht werden. Der Bedienfokus wird normalerweise über die Maus durch das Anklicken gesetzt.

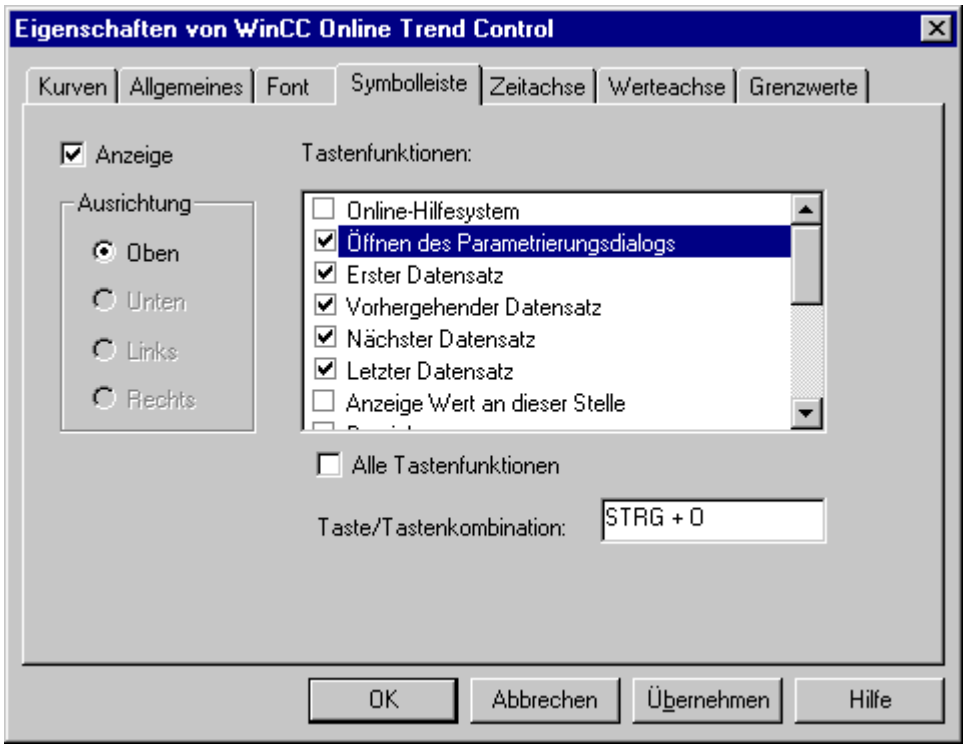
Über die Tastatur kann der Fokus über folgende, projektierbaren Wege in das Kurven- bzw. Tabellenfenster gesetzt werden:

- Wechseln des Fensters über Hotkey
- Setzen des Fokus über eine Bedientaste oder
- direktes Setzen des Fokus auf ein definiertes Element im Meldefenster bei Bildanwahl.

Die Realisierung dieser verschiedenen Varianten finden Sie in diesem Kapitel unter der Beschreibung des Alarm Logging.

Um die Bedientasten der Toolbar wie z.B. Auswahl eines Zeitbereiches zu aktivieren, kann neben der Standard-Mausbedienung auch eine Funktionstastenbedienung definiert werden. Standardmäßig sind die einzelnen Tasten mit den Funktionstasten F1 bis F10 belegt.

Für jede in der Toolbar eingeblendete Taste (*Graphics Designer* → **D** *WinCC Alarm Control* → *Symbolleiste*) kann eine eigene zugehörige Tastenbedienung in den Eigenschaften hinterlegt werden.



Über diese Projektierungsschritte können alle eingesetzten Bedientasten für das Kurven- oder Tabellenfenster mit einer Tastenkombination versehen werden. Eine Tastenbedienung für die Kurven- bzw. Tabellenfenster ist daher zusätzlich zu definieren.

In dem Kurvenfenster können folgende Standardtasten nach dem Aktivieren der jeweiligen Funktionstaste eingesetzt werden:

Bewegung	**Standard-Tasten**	**Tasteneinstellungen**
Leselineal	Richtungstasten	Leselinie nach links bzw. rechts bewegen
Zoomen	Zoomausschnitt wählen	Ersatz-Eingabehilfe für Maus einstellen und aktivieren (siehe Systemeinstellungen) EINFG und Richtungstasten ermöglichen die Festlegung des Zoomausschnittes
Dialoge	Tabulatortaste	zur Bewegung zwischen den Eingabefeldern
	Richtungstasten	innerhalb der Variablenauswahl bzw. Registerauswahl bewegen
	+ Taste (- Taste)	Aufblenden bzw. Schließen des Baumes der Variablen des Archives
	Leertaste	Auswählen bzw. Auswahl zurücknehmen
	EINGABE-Taste	Dialogbox bestätigen und beenden
	ESC-Taste	Dialogbox abbrechen.

Tag Logging anlagenspezifisch entworfene Toolbar-Tasten

Die gesamten Toolbar-Tasten sind von WinCC vorgegeben und können in der Gestalt nicht verändert werden. Wird für die zu projektierende Anlage ein vorgegebenes Layout der Tasten vorgeschrieben, so muß die WinCC Toolbar abgeblendet werden und die zugehörigen Tasten selbst entworfen werden. Alle diese neuen Tastenobjekte können nach den Wunschvorstellungen des Kunden z.B. mit Bildern versehen werden.
Die Funktionalität zu den einzelnen Tasten muß aber noch als zugehörige Aktion projektiert werden. In der *C-Aktion* des zugehörigen Ereignisses (z.B. Taste drücken) muß die entsprechende *Standard-Funktion* aus dem Funktionsbaum ausgewählt werden.

Die für die Tastenbedienung zur Verfügung gestellten Funktionen finden Sie unter *Standard-Funktionen → taglog → toolbarbuttons*. Für jede Taste aus der Toolbar findet sich eine korrespondierende Funktion in der Liste wieder. Zum Beispiel wird das Leselineal über folgende Funktion aufgerufen:

```
TlgTrendWindowPressLinealButton(lpszWindowName);
```

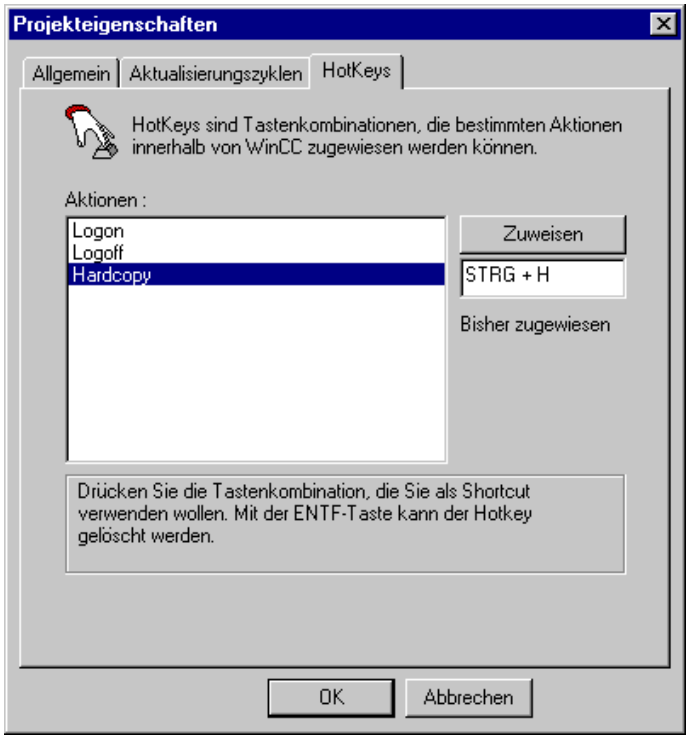
Für den Parameter muß der Fenstertitel des Tag Logging Controls eingetragen werden.

Diese Aktionen können auch für selbstentworfene Bedientasten über Mausbedienung genutzt werden.

3.3.10.5 Druckauftrag anstoßen

Ein Druckauftrag kann über mehrere Wege angestoßen werden. Im *WinCC Explorer* zum Beispiel wird direkt über die Auswahl in der Liste der Druckaufträge der Druck aktiviert. Im Anlagenbild selbst kann aber auch eine Taste kreiert werden, um den Druckauftrag über die Taste anzustoßen.
Für die Meldelisten existiert diese Taste bereits in der Toolbar und kann, wie auf den vorherigen Seiten beschrieben, mit einer Funktionstaste oder einer eigens entworfenen Taste aktiviert werden.

Ein Bildschirmabzug - die sogenannte Hardcopy - ist mittels einer Kurzbedienung in jedem Bild aktivierbar. Dieser Hotkey wird in Projekteigenschaften global eingestellt. Rufen Sie dazu im *WinCC Explorer* die *Projekteigenschaften → Hotkey → Hardcopy* auf und definieren Sie durch direkte Eingabe der Tastenkombination (Drücken der jeweiligen Taste auf der Tastatur) die Schnellbedienung.



Wird eine eigens entworfene Taste für den Druck eines definierten Druckauftrages im Anlagenbild projektiert, so muß der Anstoß über eine *C-Aktion* erfolgen. Die Taste wird wie am Anfang des Kapitels beschrieben, z.B. über eine Funktionstaste (siehe Hotkey) oder über die Tastaturbedienung (z.B. Taste D) bedient. Die *C-Aktion* muß dementsprechend an dem zugehörigen Ereignis (z.B. *Mausklick* oder *Tastatur - drücken*) projektiert werden.
Von WinCC wird diese Funktionalität mit den Funktionen in *Standard-Funktionen → Report → ReportJob* zur Verfügung gestellt.

```
ReportJob(pszJobName,pszMethodName);
```

Die Funktion erhält als ersten Parameter den Namen des Druckauftrages. Als zweiter Parameter ist "PRINT" zu übergeben, wenn sofort gedruckt werden soll, oder "PREVIEW" zu übergeben, wenn die Druckvorschau gestartet werden soll.

3.3.10.6 An- oder Abmelden

Neben den einstellbaren Hotkeys für den Login- bzw. Logout-Vorgang kann auch eine Taste projektiert werden, die das Aufblenden der Login-Dialogbox bewirkt. Das Abmelden über eine Tastenbedienung ist ebenfalls möglich. Hierzu muß jeweils eine eigene Taste entworfen werden, die zum Beispiel neben dem Mausklick auch über die Tastatur bedienbar ist. Eine Funktionstastenbedienung ist ebenfalls über die Hotkey-Eigenschaft der Taste einstellbar. Die verschiedenen Varianten für eine Tastenbedienung werden am Anfang des Kapitels im Detail beschrieben. Die Funktion, die für das An- bzw. Abmelden eingesetzt werden muß, ist eine WinCC Applikationsfunktion. Der Einsatz dieser Funktion muß als *C-Aktion* projektiert werden. Legen Sie die *C-Aktion* zum Beispiel an das Ereignis *Mausklick* oder das Ereignis *Taste drücken*.
Für das Abmelden wird folgende Funktion eingesetzt:

```
#pragma code("USEADMIN.DLL")
#include "PWRT_API.H"
#pragma code()

PWRTLogout();
```

Das Anmelden erfolgt mit der WinCC-C-Applikationsfunktion *PWRTLogin()*.
Ein Beispiel für den Einsatz dieser Funktion wäre:

```
#pragma code("USEADMIN.DLL")
#include "PWRT_API.H"
#pragma code()

PWRTLogin('1');
```

Die eingeblendete Dialogbox kann mit den Standard-Tasten bedient werden:

Bewegung	Standard-Tasten	Tasteneinstellungen
einzelne Eingabefelder	Tabulator-Taste (vorwärts) oder UMSCHALT und Tabulator-Taste (rückwärts) Richtungstasten	Leselinie nach links bzw. rechts bewegen
Bestätigen (OK)	EINGABE-Taste	Dialogbox beenden und Eingabe bestätigen
Abbrechen	ESC-Taste	Dialogbox bzw. Eingabe abbrechen

3.3.11 Bildbausteintechnik

Die Bausteintechnik ist ein wesentlicher Ansatz für die schnelle und einfache Projektierung sowie die Wiederverwendbarkeit und Wartbarkeit von projektierten Bildkomponenten.

Eine projektierte Prozeßbox wird zum Beispiel für mehrere gleichartige Prozeßkomponenten (z.B. Ventile oder Regler) eingesetzt.

Das einmalig, projektierte Bildfenster kann nun für die im Projekt zu bedienenden und zu visualisierenden Regelbausteine nach folgenden Prinzipien weitergenutzt werden:

- Kopieren eines Bildfensters und jeweilige Neuverbindung der Variablenfelder
- Einsatz eines Bildfensters, dessen Variablenfelder beim Aufruf zugewiesen werden (indirekte Verbindung)
- Einsatz von Anwenderobjekten mit Prototypen und daraus erzeugten Objekten
- Erstellung von Prototypbildern und deren Einbindung
- Erstellung von OCX-Bildbausteinen und Einbinden als WinCC-OCX-Objekte

Vergleich der verschiedenen Techniken

Diese Techniken sind in ihrem Einsatz, der Komplexität der Projektierung und ihren Möglichkeiten sehr unterschiedlich, so daß wir zunächst einen Vergleich der Alternativen vornehmen.

Art	Vorteil	Nachteil
Kopie von Bildfenstern	einfache Vorgehensweise	alle Objektverbindungen müssen geändert werden Änderungen im Bildaufbau führen zu aufwendigen Nachbearbeitungen
Bildfenster mit indirekter Verbindung	• einmaliger Aufbau des Bildfensters mit einfachen *C-Aktionen* • Wiederverwendung ohne Kopie des Basis-Bildfensters	Änderungen im Bildaufbau führen zu aufwendigen Nachbearbeitungen
Anwender-objekte	einmaliger Aufbau des Objektes mit Verbindung über vorhandene Dynamic-Wizards	• Änderungen im Bildaufbau führen zu Nachbearbeitungen, d.h. neue Bilderzeugung • nicht zentral änderbar
Prototypbilder	• einmaliger Aufbau des Objektes • zentral änderbar	(gute) C-Kenntnisse nötig

Art	Vorteil	Nachteil
OCX	• einfaches Einbinden in die Projektierung von WinCC als Objekt im Bild • eine nachträgliche Änderung des OCX-Objektes führt zu keiner Nachbearbeitung in den erzeugten Objekten, außer bei änderungen der Objekt-Eigenschaften • sehr performant • weitere grafische Möglichkeiten • Zukauf von neuen Objekten (z.B. PCS7-Bausteine)	muß per Programmierung erstellt werden (C++,VB 5), kann nicht mittels WinCC Projektierung erstellt werden.

Werden im Projekt nur wenige, einfache Bildbausteine eingesetzt, so genügt dem Projekteur eine der ersten Varianten. Diese können ohne größere Einarbeitung umgesetzt werden.
Für einfache Objekte mit kleiner bis mittlerer Komplexität und Variablenanbindung eignet sich das Anwenderobjekt sehr gut. Bedarf dieses Objekt mehrerer vorhersehbarer Änderungen, so lohnt sich eine Einarbeitung in das Konzept der Prototypbilder.
Sind die grafischen Bausteine von komplexer Natur bzw. wird eine umfangreichere Verarbeitungsleistung benötigt, so ist die OCX-Technik zu bevorzugen. Die verfügbaren OCX-Objekte werden branchenbezogen in Zukunft immer stärker wachsen.

Im folgenden zeigen wir die verschiedenen Arten der Projektierung von Bildbausteinen und deren Verwendung in den Anlagenbildern. Sie können sich auf diesem Wege selbst ein Bild von den unterschiedlichen Varianten und deren Einsatzgebieten in Ihren Projekten machen.

3.3.11.1 Prozeßbox als Bildbaustein

Zur Darstellung der aktuellen Zustände eines Aggregates (Regler, Ventile, Motoren etc.) bzw. zur Vorgabe von Sollwerten werden in den Anlagenbildern spezifische Informationsboxen eingeblendet. Diese Prozeßboxen beinhalten typischerweise sowohl aktuelle Zustände (Istwerte) als auch Vorgabewerte, die über den privilegierten Bediener eingegeben werden können:

Erstellen der Informationsbox

Diese Informationsbox wird als Bildfenster kreiert, deren Komponenten mit den zugehörigen (Prozeß-)Variablen verbunden werden.

Schritt	Typ	Projektierung
1	Daten-strukturen	Definieren Sie die im Bildbaustein einzusetzenden Datenstrukturen über den Variablenhaushalt, z.B. Motor mit Istwert, Sollwert, Ein-/ Ausschalter:
2	Bildbaustein	Projektieren Sie über den *Graphics Designer* ein Bild, das die Aggregatszustände zeigt, z.B. Balken und E/A-Felder sowie Bedientasten. Die Größe des Bildfensters (Eigenschaft Bild-Objekt - X-Größe und Y-Größe) muß der Zielgröße des Bildfensters entsprechen.
3	Variablen definieren	Definieren Sie die (Prozeß-)Variablen im Variablenhaushalt. z.B. Motor_T01 vom (Struktur-)Datentyp Motortyp, der für die Prozeßbox verwendet wird.
4	Variablen-verbindung	Dynamisieren Sie nun die einzelnen Bildkomponenten z.B. EA-Felder, Balken etc. durch Verbindung mit den zugehörigen (Prozeß-)Variablen
5	Bildfenster	In dem Anlagenbild legen Sie ein Bildfenster-Objekt an und verbinden dieses mit dem unter Schritt 2-4 entworfenen Bildfensterinhalt über die Eigenschaft Bildfenstername.
6	Eigenschaften Einstellungen	Dieses Bildfenster-Objekt soll bei Bildaufschlag noch nicht angezeigt werden. Deshalb muß die Eigenschaft *Anzeige* statisch auf nein gesetzt werden. Das Aussehen des Bildfensters mit den Windows-Tasten sowie Titel etc. müssen in den Eigenschaften des Bildfensters ebenfalls noch festgelegt werden.
7	Aufruf des Bildfensters	Dieses Bildfenster muß z.B. über eine Bedientaste oder über die Bedienung des Aggregates selbst aufgeblendet werden. Entwerfen Sie eine Bedientaste, die mit dem Einblenden des Bildfenster-Objektes verbunden wird (z.B. über *Direktverbindung*)

Dieses Bildfensterobjekt, der Bildfensterinhalt sowie der zugehörige Aufruf des Bildfensters (Bedientaste) können bei weiteren Aggregaten in ähnlicher Form wieder eingesetzt werden. Dazu muß jeweils das Bildfenster-Objekt , der Bildbaustein sowie die Bedientaste kopiert werden. Die Referenzen müssen jedesmal angepaßt werden. Für den Kopiervorgang können das Bildfenster-Objekt sowie die Bedientaste mittels Drag&Drop in die Grafikbibliothek (z.B. Projekt-Bibliothek) gelegt werden.

Anpassen der Bildbausteine

Es müssen daher folgende, einzelne Schritte bei der Verwendung des erstellten Bildbausteins durchgeführt werden:

Schritt	Typ	Projektierung
1	Prozeß-variablen	Definieren Sie eine neue Prozeßvariable, z.B. Motor_T02 zu der definierten Datenstruktur
2	Kopie des Bildbausteins	Legen Sie eine Kopie des Bildfensterinhaltes an (Motort02.PDL) und ändern Sie alle fest hinterlegten Referenzen (z.B. statt Motor_T01.Istwert nun Motor_T02.Istwert).
3	Kopie des Bildfensters	Legen Sie eine Kopie des Bildfenster-Objektes im Ziel-Anlagenbild an (mittels Drag&Drop aus der Grafikbibliothek). Passen Sie die Referenz zum Bildfensterinhalt unter der *Eigenschaft → Bildname* an (Motor02.PDL).
4	Kopie der Bedientaste	Legen Sie eine Kopie der Bedientaste im Ziel-Anlagenbild an (mittels Drag&Drop aus der Grafikbibliothek). Passen Sie die Referenz zu dem neuen Bildfenster-Objekt in der *Direktverbindung* (*Objekt → Bildfenster2 → Anzeige)* an.

Auf diese Weise können die einzelnen Bildfenster und deren Inhalt pro Aggregat erstellt und über Kopiervorgänge wieder verwendet werden. Wie bereits sichtbar, besteht der Aufwand jeweils in der Anpassung der fest hinterlegten Referenzen bei dem Bildfensterinhalt zum Beispiel. Aus diesem Grunde kann über den Weg der indirekten Adressierung eine einfachere Wiederverwendbarkeit erreicht werden. Der Anpassungsaufwand soll auf ein Minimum reduziert werden.

Als alternative Lösung kann der Bildbaustein auch ohne Bildfensteranbindung projektiert werden. Dies bedeutet, daß der Bildbaustein selbst als nicht angezeigtes Objekt im Anlagenbild projektiert wird. Dies hat aber bei der Änderung des Bildbausteines den erheblichen Nachteil, daß in allen Bildern, in denen dieser Bildbaustein verwendet wird, jeweils eine Änderung durchgeführt werden muß.

3.3.11.2 Bildbaustein mit indirekter Adressierung

Bisher wurden die einzelnen Komponenten des Bildbausteins fest mit den zugehörigen (Prozeß-) Variablen verbunden. Wird die Verbindung nicht durch eine fixe Projektierung, sondern dynamisch zur Laufzeit ermittelt, kann der entworfene Bildbaustein wesentlich flexibler eingesetzt werden. Diese dynamische Verbindung von (Prozeß-) Variablen wird über die indirekte Adressierung der einzelnen Komponenten im Bildbaustein realisiert. Dies bedeutet, daß nicht direkt mit der (Prozeß-) Variablen verbunden wird, sondern nur mit dem *Container*, der den aktuellen Namen der zugehörigen (Prozeß-) Variablen zur Laufzeit tragen wird.

Der Anpassungs- sowie Wiederverwendbarkeitscharakter eines Bildbausteins kann damit wesentlich vereinfacht werden.
Die Projektierung wird ähnlich der Schritte wie zuvor beschrieben durchgeführt. Die Schritte im einzelnen:

Schritt	Typ	Projektierung
1	Daten-festlegung	Definieren der im Bildbaustein einzusetzenden Daten über den Variablenhaushalt, z.B. Motor001_ Istwert, Motor001_Sollwert, Motor001_Schalter einerseits und Festlegung der Namens-Container für die einzelnen Komponenten, die im Bildbaustein verwendet werden sollen, z.B. Istw_Name, Sollw_Name, etc. andererseits. Diese Variablen initialisieren Sie mit einem Namen, z.B. Motor001_Sollwert.
2	Bildbaustein	Projektieren Sie über den *Graphics Designer* ein Bild, das die Aggregatszustände zeigt, z.B. Balken und E/A-Felder sowie Bedientasten. Die Größe des Bildfensters (Eigenschaft Bild-Objekt - X-Größe und Y-Größe) muß der Zielgröße des Bildfensters entsprechen.
3	Variablen-verbindung	Dynamisieren Sie nun die einzelnen Bildkomponenten z.B. EA-Felder, Balken etc. mit Verbindung der zugehörigen Container-Variablen, die den Namen der entsprechenden Variablen enthalten. Bei der Verbindung muß aber nun hinterlegt werden, daß die Variable nur der Name der eigentlichen (Prozeß-) Variablen ist. Deshalb muß jeweils in der Spalte *indirek.Adr.* angekreuzt werden.
4	Bildfenster	In dem Anlagenbild legen Sie ein Bildfenster-Objekt an und verbinden dieses mit dem unter Schritt 2-3 entworfenen Bildfensterinhalt über die Eigenschaft Bildfenstername.
5	Eigenschaften-Einstellungen	Dieses Bildfenster-Objekt soll bei Bildaufschlag noch nicht angezeigt werden. Deshalb muß die Eigenschaft *Anzeige* statisch auf nein gesetzt werden. Das Aussehen des Bildfensters mit den Windows-Tasten sowie Titel etc. müssen in den Eigenschaften des Bildfensters ebenfalls noch festgelegt werden.
6	Aufruf des Bildfensters	Dieses Bildfenster muß z.B. über eine Bedientaste oder über die Bedienung des Aggregates selbst aufgeblendet werden. Entwerfen Sie eine Bedientaste, die mit dem Einblenden des Bildfenster-Objektes verbunden wird (z.B. über *Direktverbindung*).
7	Grafik-bibliothek	Das Bildfenster-Objekt sowie die Bedientaste werden mittels Drag&Drop zur Wiederverwendung in die Bibliothek gelegt.

3.3.11.3 Anwenderobjekte

Mittels Anwenderobjekten und den zugehörigen Dynamic-Wizards können Bildbausteine erstellt werden, die einfach wiederverwendet werden können. Die erstellte Kopie des Bildbausteines kann mit einer einfachen Wizard-Projektierung mit den zugehörigen, aktuellen (Prozeß-)Variablen verbunden werden.
Ein Anwenderobjekt ist ein vom Projekteur entworfenes grafisches Objekt (z.B. Kombination mehrerer), dessen viele Eigenschaften und Ereignisse mittels eines Konfigurationsdialoges auf die wesentlichen Eigenschaften und Eigenschaften beschränkt werden. Dieses Anwenderobjekt wird mittels des zugehörigen Wizard als Prototyp dynamisiert.

Im einzelnen sind folgende Schritte notwendig:

Schritt	Typ	Projektierung
1	Daten-strukturen	Definieren der im Bildbaustein einzusetzenden Datenstrukturen über den Variablenhaushalt.
2	Bildbaustein	Projektieren Sie über den *Graphics Designer* ein Anwenderobjekt mit den benutzerdefinierten Eigenschaften

Ein Anwenderobjekt wird aus einer Gruppe von WinCC-Objekten gebildet. An diesen Objekten ist zunächst keine Dynamik projektiert. Alle Objekte, die zu dem Anwenderobjekt zusammengefaßt werden sollen, werden selektiert und der Konfigurationsdialog des Anwenderobjektes wird aufgerufen:

In diesem Dialog werden nun alle Eigenschaften der Objekte zur Eigenschaft des Anwenderobjektes erklärt, die später dynamisiert werden sollen. Die Basis-Eigenschaften für ein Objekt (z.B. Position und Größe) wurden bereits für das Anwenderobjekt hinterlegt. Jede einzelne Eigenschaft der zusammengefaßten Objekte kann im Dialog ausgewählt werden und per Drag&Drop als *Benutzerdefinierte Eigenschaft bzw. Ereignis* dem neuen Anwenderobjekt hinzugefügt werden.

Jeder dieser Eigenschaften kann ein neuer (sprachenunabhängiger) Attributname vom Anwender vergeben werden sowie auch der sprachenabhängige Eigenschaftsname (z.B. bei englischer Projektierung). Eigenschaften, die im Eigenschaftendialog nicht sichtbar dargestellt werden sollen, aber z.B. bei Scripten genutzt werden, können mittels des Zeichens @ verborgen werden. Damit können nur wenige (zu dynamisierenden) Eigenschaften und Ereignisse nach außen geführt werden. Alle anderen werden verborgen.

Das entworfene Anwenderobjekt muß nun dynamisiert werden. Dazu wird ein Wizard zur Verfügung gestellt:

Schritt	Typ	Projektierung
3	Dynamisieren	Rufen Sie den Dynamic-Wizard *Prototyp dynamisieren* auf. Verbinden Sie schablonenmäßig (d.h. prototypisch) jede einzelne Eigenschaft des Objektes mit der zugehörigen Strukturkomponente der definierten Datenstruktur. Über den Variablen-Browser wird der *Strukturmember* zur Verschaltung ausgewählt. Der Wizard speichert aber nur den Namen der Strukturkomponente an der verbundenen Eigenschaft (z.B. .Value). Es muß jede einzelne Eigenschaft separat verbunden werden. Dieses Objekt ist nun ein dynamisiertes Objekt, das aber nur prototypisch verbunden wurde und zur Laufzeit noch nicht lebt. Es kann daher zur Laufzeit nicht aktualisiert werden.
4	Ablegen in die Grafik-bibliothek	Legen Sie diesen Prototyp als Objekt in die Grafikbibliothek

Das Prototyp-Anwenderobjekt wird z.B. für die mehrfache Wiederverwendung in die Grafikbibliothek gelegt. Ein Beispiel für ein dynamisches Objekt sind die Zeigerinstrumente in der WinCC-Bibliothek (Anwenderbibliothek, Anwenderobjekte, Zeigerinstrumente).

Das Prototyp-Objekt wird in dem Ziel-Anlagenbild als Kopie des Prototyps eingehängt. Nun muß diese Kopie mit den echten (Prozeß-) Variablen aus dem Variablenhaushalt verbunden werden.

Schritt	Typ	Projektierung
5	Variable	Definieren Sie zu der im Schritt 1 definierten Datenstruktur eine (Prozeß-) Variable, die für das Anwenderobjekt verwendet werden soll.
6	Instanz anlegen	Kopieren Sie das Prototyp-Objekt aus der Grafikbibliothek mittels Drag&Drop in das Anlagenbild. Verbinden Sie dieses Objekt mit der (Prozeß-) Variablen über den Dynamic-Wizard Prototyp fest instanzieren: Der Wizard verbindet automatisch alle benötigten Strukturkomponenten der Variablen mit der richtigen Eigenschaft des prototypischen Bildbausteins, indem jede prototypische Variablenanbindung an jeder Eigenschaft durch die konkrete Variablenanbindung ausgetauscht wird. Nun ist ein Objekt entstanden, das zur Laufzeit mit den aktuellen Variablenwerten aktualisiert wird.

3.3.11.4 Dynamische Instanz

Neben dem Dynamic-Wizard *Prototyp fest instanzieren* existiert auch noch ein Wizard für *Prototyp dynamisch instanzieren*. Was ist der Unterschied zu fest instanzieren und welche Schritte müssen modifiziert werden?
Im Gegensatz zum festen Verbinden eines Objektes mit Variablen können die Bildbausteine auch dynamisch verbunden werden. Dies bedeutet, daß abhängig von dem aktuellen Inhalt einer Variable die Instanz zur Laufzeit erst gesetzt wird. Zum Beispiel wird der obige Bildbaustein nicht fest mit der Variable verbunden, sondern es wird der Name der Variable dynamisch gehalten. Der aktuelle Name der Variable muß dann über eine Textvariable ermittelt werden. Diese Textvariable, die den jeweils aktuellen Namen der Variablen enthält, muß mit dem Bildbaustein verbunden werden.
Im Gegensatz zur festen Instanzierung müssen bei der Projektierung folgende Schritte geändert werden:

Schritt	Typ	Projektierung
2	Bildbaustein	Projektieren Sie über den *Graphics Designer* ein Anwenderobjekt mit den benutzerdefinierten Eigenschaften wie oben beschrieben. Das Anwenderobjekt muß eine Komponente *Statischen Text* beinhalten, dessen Eigenschaft *Text* als benutzerdefinierte Komponente übernommen wird. Diese Eigenschaft Text erhält als Attributnamen *Tagname*. Dieser Tagname wird für die dynamische Anbindung der (Prozeß-)Variablen verwendet.
5	Variable	Definieren Sie zu der im Schritt 1 definierten Datenstruktur eine (Prozeß-) Variable, die für das Anwenderobjekt verwendet werden soll.
6	Instanz anlegen	Kopieren Sie das Prototyp-Objekt aus der Grafikbibliothek mittels Drag&Drop in das Anlagenbild. Verbinden Sie dieses Objekt mit der (Prozeß-) Variablen über den Dynamic-Wizard Prototyp fest instanzieren: Der Wizard verbindet automatisch alle benötigten Strukturkomponenten der Variablen mit der richtigen Eigenschaft des prototypischen Bildbausteins, indem jede prototypische Variablenanbindung an jeder Eigenschaft durch die konkrete Variablenanbindung ausgetauscht wird. Nun ist ein Objekt entstanden, das zur Laufzeit mit den aktuellen Variablenwerten aktualisiert wird.

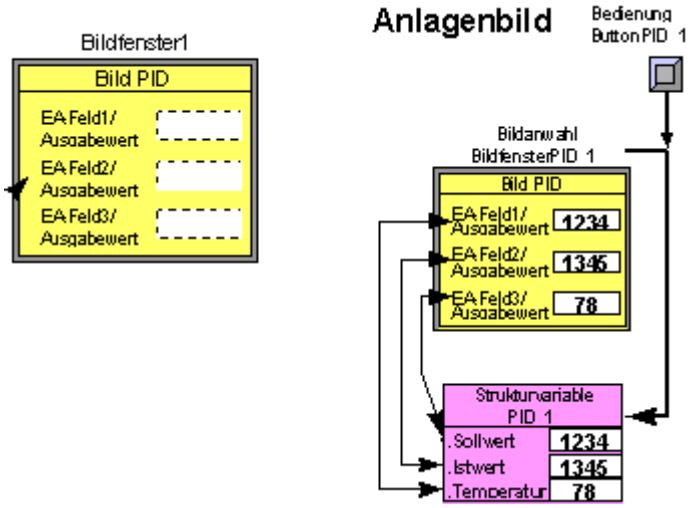
Beim Einsatz der dynamisierten Anwenderobjekte und Prototypen ist von seiten des Projekteurs darauf zu achten, daß die nötigen *C-Aktionen* bereits an den Objekten hinterlegt wurden. Diese dürfen nicht gelöscht werden, da sonst die gesamte Baustein-Funktionalität verlorengeht.

3.3.11.5 Prototypbilder

Die Technik der Prototypbilder geht noch einen Schritt weiter. Beim Einsatz von Prototypen kann das Konzept so flexibel aufgebaut werden, daß eine Änderung an dem Prototyp automatisch eine Nachführung an den erzeugten Objekten erfolgt. Diese Flexibilität erfordert sehr flexible C-Scripte, die angebunden werden müssen.
Die Technik der Prototypbilder arbeitet mit sogenannten Templatebildern, die mehrfach in ein oder mehrere Vaterbilder eingebunden werden. Ein Templatebild ist nur eine Schablone, die erst in einem echten Objekt zum Leben erweckt wird. Ein Objekt auf Basis einer Schablone (=Prototypbild) entsteht durch eine sogenannte Instanzierung. Es können mehrere Instanzen (d.h. echte Objekte) zu einer Schablone erstellt werden.

Bildfenster Schablone

Eingesetztes Bildfenster mit aktuellen Daten aus dem Datenmanager angzeigen.



Nachträgliche Änderungen finden an zentraler Stelle (in der Schablone) statt und wirken sich auf alle Anwendungen (Instanzen) aus. Damit ist dies eine sehr effiziente Technik, die ein mühsames Nachziehen von Änderungen an vielen Stellen erübrigt.

In einem Vaterbild können bis zu 30 Instanzen (d.h. Objekte) jeweils eines Templatetyps eingeblendet werden. Beim Einsatz unterschiedlicher Prototypen können auch mehr als 30 Objekte eingesetzt werden.

Die Prototypbilder sind Bildbausteine, die nach der Erstellung in die Bibliothek zur Wiederverwendung gelegt werden. Die eingesetzten Bildbausteine werden als Instanzen der Schablone in den Anlagenbildern verwendet. Diese Kopien zeigen die jeweils aktuellen Daten z.B. von Reglern oder Motoren, die in dem Baustein visualisiert werden. Die Darstellung der zugehörigen Regler- oder Motorkomponenten erfolgt automatisch.

Es können einfache, aber auch komplexe Bildbausteine gebildet werden. Ein Bildbaustein kann aus mehreren Komponenten bestehen. Diese sind teilweise oder ganz überlappt sind, aber eine zusammengehörige Einheit bilden. So können zum Beispiel alle Daten zu einem Motor, wie die Sicht auf den aktuellen Zustand, Verlaufsdaten, Wartungsdaten, etc. in einem Objekt zusammengefaßt und je nach Bedarf aktualisiert werden. Sind viele Motoren desselben Typs vorhanden, brauchen nach dem einmaligen Erstellen des Bildbausteins nur noch Kopien gebildet werden. Alles andere läuft automatisch.

Zunächst betrachten wir die Erstellung der Schablone (Templatebild).

Schritt	Typ	Projektierung
1	Daten-strukturen	Anlegen einer Variablenstruktur (Datentyp Strukturtyp) im Datenmanager; hierbei wird festgelegt, aus wievielen Variablen die Struktur aufgebaut ist (Membervariablen), wie diese heißen und von welchem Datentyp diese jeweils sind (BIT, SHORT, etc.). z.B. PID mit Sollwert, Istwert und Temperatur als Sturkutkomponenten.
2	Bildfenster-aufbau	Erstellen eines Bildes, das als Bildbaustein genutzt werden soll. Typischerweise ist dies kleiner als die Bildschirmgröße und kann entsprechend dimensioniert werden. Jedes Bild, das einmal entstanden ist, kann für die Bildung eines Templates verwendet werden. Das Bild wird mit den Grafikeditierfunktionen erstellt und es werden darin Grafikvariablen, wie EA-Felder, Balken, Zustandsanzeigen etc. Plaziert, aber nicht mit Variablen verbunden. Interne Beziehungen (*Direktverbindung*) zwischen Grafikobjekten, wie z.B. die änderungsgesteuerte Übergabe des Ausgabewertes eines EA-Feld an einen Balken, werden in diesem Bild projektiert.
3	Datenstruktur und Bild-fensteraufbau verbinden	Nun werden die Grafikfelder mit den Strukturkomponenten der zugehörigen Variablenstruktur verbunden. Diese Verbindung-projektierung verbindet auf Typebene (nur Schablone) und noch nicht mit konkreten Pozeßobjekten. Zu diesem Zweck gibt es ein vorbereitetes Beispiel-Projekt auf der WinCC CD (\Samples). In der Projektbibliothek (\Template) dieses Projektes gibt es ein Anwenderobjekt *TemplateInit*, das die Verschaltung vornimmt. Es liegt in der Grafikbibliothek und kann von dort in ein Bild, das typisiert werden soll, per Drag&Drop gezogen werden. *TemplateInit* verfügt bereits über eine fertige Scriptlogik. Sie arbeitet mit einer sogenannten *ConnectionTable*, die in der Projektierung als Tabelle ausgefüllt wird und genau die vor-genannten, unterstrichenen Einträge enthält. Über diesen Weg wird die Verbindung zwischen den Eigenschaften und den Strukturkomponenten festgelegt. Der Aufbau dieser Verbindungen kann innerhalb eines Templates oder auch von außerhalb gesetzt werden. In der speziellen Projekt-Grafikbibliothek liegen hierzu Anwenderobjekte, die rein äußerlich wie Bedientasten aussehen, die aber parametrierbare Information über die aufzurufende Schablone halten. Die gesamten Scripte für die Realisierung dieser bereits vorbereiteten Prototyperstellung müssen in das Ziel-Projekt geladen werden. Siehe dazu die abschließenden Schritte am Ende des Abschnittes unter Schritt 8-10. Ohne diese Projektfunktionen sind diese Prototypen nicht realisierbar.

Schritt	Typ	Projektierung
4	Bildfenster	Dieses Bild soll als Prozeßbox verwendet werden. Erstellen Sie dazu in einem temporären Bild (d.h. es wird nur für diesen Zwischenschritt benötigt) ein Bildfenster-Objekt und verbinden Sie die Eigenschaft *Bildname* dieses Objektes mit dem Bild, das den Bildbaustein enthält.
5	Grafik-bibliothek	Legen Sie dieses Bildfenster-Objekt mittels Drag&Drop in die Grafikbibliothek. Nun kann der Bildbaustein in den Anlagenbildern über das Holen aus der Grafikbibliothek eingesetzt werden.

Diese Schablone kann nun mehrmals in den Anlagenbildern eingesetzt werden. Die Verbindung zu den Prozeßvariablen erfolgt automatisch über die Namensgebung.

Schritt	Typ	Projektierung
6	Variable definieren	Definieren Sie eine (Prozeß-) Variable zu dem zugehörigen Datentyp (z.B. PID_1 vom Typ PID).
7	Instanz anlegen	Kopieren Sie den Bildfenster-Baustein aus der Grafikbibliothek Vergeben Sie den Bildfenster-Objektnamen gleich dem Namen der (Prozeß-) Variablen, die Verwendung finden soll (z.B. PID_1): *Bildfensterobjekt* → *Bildfenster* → *Objektname* → *Statik* **D** auf PID_1 setzen

Beim Plazieren erhält der Bildbaustein den Namen einer strukurierten (Prozeß-) Variable, deren Werte die Zustandsdaten eines Prozeßobjektes enthalten. Zur Laufzeit versorgt sich der Bildbaustein dann automatisch mit den Zustandsdaten.

Für diese Prototypbilder werden einige C-Scripte verwendet, die als Projektfunktionen abgelegt werden bzw. wurden. Um die bereits vorbereiteten C-Scripte verwenden zu können, müssen folgende Scripte aus dem Beispielprojekt übernommen werden. Gehen Sie wie folgt vor:

Schritt	Typ	Projektierung
8	Kopieren der Funktions-dateien	Kopieren Sie aus dem Beispielsprojekt vom Pfad <Projektpfad>\Library alle benötigten Funktionen (.fct), z.B. LinkConnectionTable.fct in Ihr Projektverzeichnis <Projektpfad>\Library.
9	im Projekt bekanntgeben	Rufen Sie in WinCC den Global Script Editor (*Global Script* → ***R*** → *Öffnen*) auf, um diese neuen Funktion im folgenden bekanntzugeben: über die Bedientaste *Header generieren* können Sie nun die neuen Funktionen im Funktionsbaum der Projektfunktionen bekanntgeben. Die neuen Funktionen sind nun in der Liste der Projektfunktionen sichtbar.
10	Anwender-objekte übernehmen	Die Anwenderobjekte werden über die Projektbibliothek zur Verfügung gestellt. In einem neuen Projekt, in dem Sie bisher keine eigenen Symbole in der Bibliothek abgelegt haben, können Sie die Bibliothek aus dem Beispielprojekt einfach in Ihr Projekt kopieren: <Projektpfad>\Library\library.pxl → in Ihren Pfad <Projektpfad>\Library kopieren! Ansonsten übertragen Sie die Anwenderobjekte über den Export-Mechanismus von einem Projekt in das andere Projekt: Exportieren Sie die gewünschten Symbole im Beispielprojekt als .emf-Dateien (*Datei → Export)* und importieren Sie diese .emf-Symbole im eigenen Projekt durch *Einfügen → Import* in ein temporäres Bild. Übertragen Sie die Symbole in Ihre Projekt-Bibliothek mittels Drag&Drop. Nutzen Sie dazu auch eine eigene Mappe z.B. Template.

Bildbausteine (Typisierte Bildanteile oder Templates) bieten einen sehr großen Einsparungseffekt. Sie sind als Objekte mit Variablenreferenzen modelliert und werden z.B. in die Grafikbibliothek abgelegt. Sie werden aus dieser Grafikbibliothek entnommen, im Anlagenbild plaziert und zur Laufzeit automatisch mit Daten versorgt. Eine Verbindungsprojektierung von einzelnen Variablen mit Grafikanteilen, wie E/A-Felder, Balken, etc. ist nicht mehr notwendig.

Beim Einsetzen dieser Prototypbilder gibt es unterschiedliche Wege, wodurch die Bildbaustein-Komponenten mit den aktuellen Namen versorgt werden. Dies erfolgt über folgende Varianten, die im folgenden kurz genannt werden sollen:

- der Instanzname wird bei **Anwahl** des Bildfensters selbst ermittelt:
  dazu wird eine vorgegebene Projektfunktion (EnabelTemplateInstance) bei dem Ereignis Bildanwahl im Bildfenster selbst hinterlegt.
- Es wird der Instanzname über eine **Ein-/Ausgabevariable** festgelegt, die automatisch über ein Script im Bildfenster gelesen und der Instanzname somit ermittelt wird,
  hierzu wird das vorbereitete Anwenderobjekt *InstanceCallButton+Template* eingesetzt.
- Eine **Bedientaste** übergibt den Instanznamen direkt an das aufgerufene Bildfenster:
  hierzu wird das vorbereitete Anwenderobjekt *InstanceCallButtons+Template* eingesetzt.

Zu diesen Varianten finden Sie in dem Beispielsprojekt ausführliche Beispiele auf der WinCC CD (\Samples).

3.3.11.6 OCX-Objekte

OCX- oder ActiveX - Objekte sind Bildbausteine, die als ablauffähige Komponenten zur Verfügung stehen. Standardmäßig werden einige von WinCC angeboten, z.B. die WinCC Digital/Analog Clock Control.

Diese Bausteine können sehr einfach in die Anlagenbilder eingebracht werden.

Schritt	Typ	Projektierung
1	OCX-Objekt einhängen	Wählen Sie in der Objektpalette im *Graphics Designer* unter den Smart-Objekten den Typ *OLE-Control* (OCX) aus. Ziehen Sie im Anlagenbild das Objekt mittels gedrückter auf und wählen Sie in der anschließenden Dialogbox das gewünschte Element aus.
2	Eigenschaften verbinden	Das eingehängte Objekt hat ebenfalls Eigenschaften und Ereignisse. Welche Eigenschaften bzw. Ereignisse zur Verfügung stehen sind von dem spezifischen OCX selbst abhängig. Eine Verbindung mit einer Variablen kann z.B. mit der Eigenschaft Prozeßanschluß vorgenommen. werden.

Erstellung

Die OCX-Bildbausteine müssen mit einer eigenen Entwicklungsumgebung erstellt werden. Dies ist zum Beispiel Microsoft Visual C++ 5 oder Microsoft Visual Basic 5.
Über diesen Weg werden die Bildbausteine geändert bzw. verbessert. Die OCX-Bausteine sind sehr performant, können aber mit WinCC Projektierungsmitteln nicht erstellt werden. In diesem Fall ist man immer auf eine externe Erstellung bzw. Änderung angewiesen.
Die Technik der Prototypbilder kommt im Gegensatz zu der änderungsfreundlichen und performanten OCX-Programmierung mit reinen WinCC-Mitteln aus. Das heißt, das Know how um die OCX-Programmierung ist dabei nicht notwendig.

Es werden heute bereits eine Vielzahl an derartigen Bausteinen angeboten. Unter anderem werden in Zusammenhang mit der Integration der beiden Welten Bedienen&Beobachten als auch der SPS-Programmierung (PCS7) im Anlagenbereich fertige Bausteine als PCS7-Faceplates angeboten.

Registrierung

Die erstellten bzw. zugekauften OCX-Bausteine müssen auf der jeweiligen WinCC-Station registriert werden. Welche OCX-Objekte auf der WinCC-Station verfügbar sind, kann bereits im Auswahl-Dialog im *Graphics Designer* (siehe obige Beschreibung) ersehen werden. Alle auf der Maschine registrierten OCX-Elemente werden in dem Dialog aufgeführt. Ein OCX-Element ist in Form einer Datei mit der Extension .OCX oder .dll auf der Maschine abgelegt.
Wurde ein Baustein noch nicht registriert, so kann dies in dem OLE-Control-Dialog von WinCC ebenfalls erfolgen. Die Dialog-Box enthält eine Bedientaste für das Registrieren als auch für das Entfernen der Registrierung der aktuell angewählten Komponente.
Für den Registriervorgang muß die zugehörige Datei auf der WinCC-Station vorliegen.

Die Verträglichkeit bzw. die Funktionsweise der OCX-Komponenten muß von dem Projekteur selbst getestet werden. Nur die mit WinCC gekennzeichneten OCX-Bausteine sind in der WinCC-Umgebung eingesetzt und getestet worden.

3.3.12 Online Projektierung (Runtime) - Hinweise, Einschränkungen

Bei der online Projektierung gibt es einige Dinge zu beachten.
Aus verschiedenen Gründen können einige wenige Änderungen online nicht oder nur unter bestimmten Voraussetzungen durchgeführt werden oder die Änderungen werden erst zu einem späteren Zeitpunkt wirksam.

WinCC Explorer

Folgende Änderungen werden nicht übernommen:

- Änderung des Typs eines Rechners in der Rechnerliste

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- Löschen/Umbenennen von Variablen
- Änderung des Datentyps einer Variablen

Alarm Logging

Folgende Änderungen werden nicht übernommen:

- Änderung der Archive / Protokolle
- Änderung der Sammelmeldungen
- Jede Meldung die über eine gesamte Summe von 500 Einzelmeldungen bei laufendem Runtime

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen

Tag Logging

Folgende Änderungen werden nicht übernommen:

- keine Einschränkungen.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- Tabellen der Anwenderarchive können angelegt, aber nicht verändert werden.
- Das Löschen von Daten in Tag Logging und Anwenderarchiven.

Ausnahmen bei der Projektierung im Runtime:

- Über das Runtime-API von Tag Logging können die Tabellen der Anwenderarchive bearbeitet und gelöscht werden.

Global Script

Folgende Änderungen werden nicht übernommen:

- Die Änderung eines Wizard Scriptes werden erst nach Neustart vom *Graphics Designer* übernommen.
- Geänderte Wizard Scripte.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen

Report Designer

Folgende Änderungen werden nicht übernommen:

- Änderungen am Meldefolgeprotokoll, da es im Runtime einmal gestartet immer aktiv bleibt und die Layoutinformationen nicht wieder neu lädt.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen.

Redundancey

Folgende Änderungen werden nicht übernommen:

- Der Rechnername des Partners kann nicht auf einen dritten Rechner umgeschaltet werden.
- Der AutoSwitcher kann nicht geändert werden, d.h. man muß am Anfang projektieren, wohin der AutoSwitcher umschalten soll. Er schaltet allerdings auch zurück falls der andere ausfällt.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen

SIMATIC S7 Protokoll Suite oder S/7PMC Kanal

Folgende Änderungen werden nicht übernommen:

- Alle Diagnoseparameter über die S7Chn.ini (nicht veröffentlicht) werden nicht online übernommen.
- Alle Änderungen an Kommunikationsadressen werden zwar online übernommen, werden aber nur bei einem Verbindungsaufbau ausgewertet.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen

Text Library

Folgende Änderungen werden nicht übernommen:

- keine Einschränkungen.
  - In der Text Library werden die geänderten Texte mit *Datei → Änderungen an aktives Projekt senden* übernommen
  - Im Alarm Logging werden die Änderungen in die Textbibliothek mit *Datei → Speichern* übernommen.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen

User Administrator

Folgende Änderungen werden nicht übernommen:

- Änderungen der Benutzerberechtigung werden erst nach erneutem Login bzw. Logoff wirksam.

Im Runtime sind folgende Projektierungsschritte nicht möglich:

- keine Einschränkungen

4 WinCC C-Kurs

Zur Dynamisierung von Objekten stehen in WinCC verschiedene Möglichkeiten zur Verfügung. Dies sind unter anderem *Variablenanbindungen*, *Dynamik-Dialoge* und *Direktverbindungen*. Mit diesen Hilfsmitteln ist die Realisierung komplexer Dynamisierungen möglich. Trotzdem stoßen diese mit wachsenden Anforderungen an ihre Grenzen. Ein viel größeres Spektrum an Möglichkeiten stehen dem Anwender durch die Projektierung von *C-Aktionen*, *Projekt-Funktionen* oder *Aktionen* zur Verfügung. Diese sind in der WinCC Scriptsprache C zu erstellen. Für viele Anwendungen ist es nicht notwendig, über umfangreiche Kenntnisse in C zu verfügen. Es ist ausreichend, wenn vorhandene Funktionen mit Parametern versorgt werden. Um jedoch die gesamte Leistungsfähigkeit von C als WinCC Scriptsprache nutzen zu können, sind einige grundlegende Kenntnisse dieser Programmiersprache notwendig. Dieser Kurs kann ihnen diese Kenntnisse vermitteln.

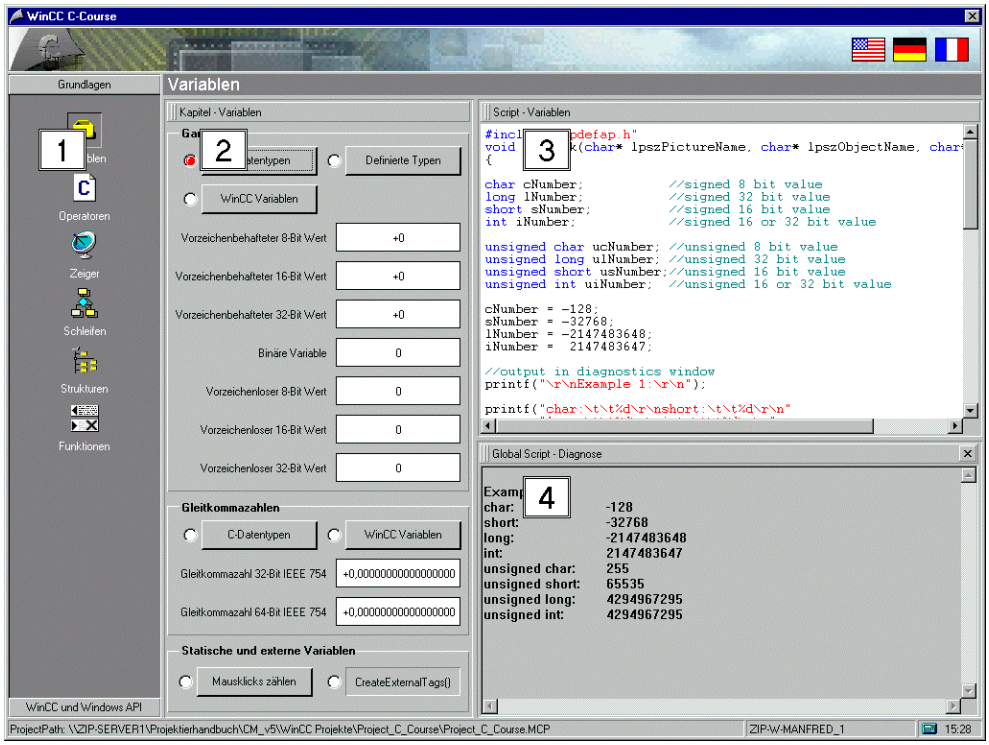
Zielgruppe

All denen, die mit C nicht vertraut sind, soll dieser Kurs grundlegende Kenntnisse im allgemeinen Umgang mit der Programmiersprache C vermitteln. Erfahrenen C Programmieren kann dieser Kurs die Besonderheiten des Einsatzes von C in WinCC näher bringen.

Das Beispielprojekt für diesen Kurs kann direkt aus dem Onlinedokument auf die Festplatte kopiert werden. Dieses wird standardmäßig im Verzeichnis *C:\Configuration_Manual* abgelegt.


Project_C_Course

Beispielprojekt



Die Oberfläche des Beispielprojektes ist in mehrere Bereiche unterteilt. Diese sind nachfolgend aufgelistet.

- **Navigationsleiste (1)**: Über die Navigationsleiste können die den verschiedenen Kapiteln zugeordneten Bilder angewählt werden.
- **Kapitelfenster (2)**: Im Kapitelfenster werden die den einzelnen Kapiteln zugeordneten Bilder angezeigt. Diese Bilder enthalten alle in den jeweiligen Kapiteln beschriebenen Beispiele. Die Beispiele sind zum Großteil an Buttons erstellt.
- **Scriptfenster (3)**: Im Scriptfenster wird der Code zum aktuell im Kapitelfenster selektierten Beispiel angezeigt. Das aktuell selektierte Beispiel ist im Kapitelfenster durch eine rote Markierung gekennzeichnet.
- **Diagnosefenster (4)**: Im Diagnosefenster werden die in den verschiedenen Beispielen über die Funktion *printf()* gemachten Ausgaben angezeigt.

4.1 Entwicklungsumgebung für C Scripte

Zur Erstellung von C Scripten stehen in WinCC zwei verschiedene Editoren zur Verfügung. Es ist dies zum Einen der Aktionseditor im Graphics Designer zur Erstellung von *C-Aktionen* an Objekten. Zum Anderen ist dies der Editor Global Script zur Erstellung von Projekt-Funktionen sowie globalen Aktionen. Die Syntax der Scriptsprache entspricht Standard C genormt nach ANSI.

Ein weiteres Einsatzgebiet der Programmiersprache C in WinCC besteht in der Erstellung von *Dynamik Wizards*. Dazu steht ebenfalls ein eigener Editor zur Verfügung. Der Umgang mit diesem Editor wird im Rahmen des Beispiels zum *Dynamik Wizard* erläutert und in diesem allgemeinen Überblick nicht berücksichtigt.

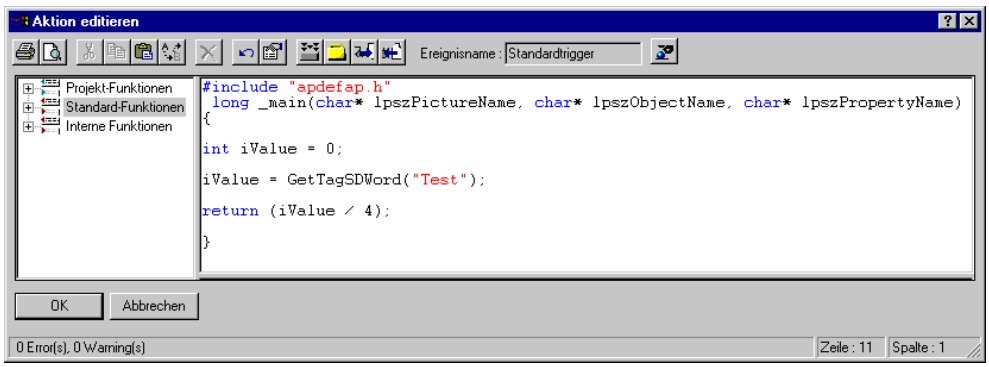
4.1.1 Aktionseditor des Graphics Designer

Im *Graphics Designer* können Objekteigenschaften über *C-Aktionen* dynamisiert werden. Ebenso kann mit *C-Aktionen* auf Objektereignisse reagiert werden.

Aktionseditor

Zur Projektierung einer *C-Aktion* steht ein Aktionseditor zur Verfügung. Dieser ist im Dialog *Objekteigenschaften* über **R** auf der gewünschten *Eigenschaft* oder *Ereignis* und *C-Aktion* zu öffnen. Bereits erstellte *C-Aktionen* werden durch einen grünen Pfeil an der *Eigenschaft* oder am *Ereignis* gekennzeichnet.

Im Aktionseditor kann die *C-Aktion* programmiert werden. Für *C-Aktionen* an Eigenschaften ist ein Trigger festzulegen. Bei *C-Aktionen* an Ereignissen ist dies nicht notwendig, da hier das Ereignis selbst der Trigger ist. Die fertige programmierte *C-Aktion* ist zu übersetzen. Treten bei der Übersetzung keine Fehler auf, kann der Aktionseditor mit *OK* beendet werden.



Aufbau einer C-Aktion

Eine *C-Aktion* entspricht im Allgemeinen einer Funktion in C. Man kann zwei verschiedene Typen von *C-Aktionen* unterscheiden. Solche die an *Eigenschaften* erstellt sind und solche die an *Ereignissen* erstellt sind. Eine *C-Aktion* an einer *Eigenschaft* wird im Allgemeinen dazu verwendet, den Wert dieser *Eigenschaft* in Abhängigkeit von verschiedenen Umgebungsbedingungen (z.B. dem Wert einer Variable) zu steuern. Bei dieser Art der *C-Aktionen* ist ein Trigger festzulegen, welcher deren Ausführung kontrolliert. Eine *C-Aktion* an einem *Ereignis* wird dazu verwendet, auf dieses *Ereignis* zu reagieren.

C-Aktion an einer Eigenschaft

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

/*1*/ long lReturnValue;

/*2*/ lReturnValue = GetTagSDword("S32i_course_test_1");

/*3*/ return lReturnValue;

}
```

Das vorhergehende Codebeispiel stellt eine typische *C-Aktion* an einer *Eigenschaft* dar. Die Bedeutung der einzelnen Abschnitte ist nachfolgend beschrieben.

- **Kopf (grau)**: Die ersten drei in der Abbildung grau unterlegten Zeilen bilden den Kopf der *C-Aktion*. Dieser wird automatisch generiert und kann nicht geändert werden. Bis auf den Typ des Rückgabewertes (im vorhergehenden Codebeispiel *long*) ist der Funktionskopf für alle Eigenschaften identisch. Der *C-Aktion* werden drei Parameter übergeben. Es handelt sich dabei um den Bildnamen (*lpszPictureName*), den Objektnamen (*lpszObjectName*) und den Eigenschaftsnamen (*lpszPropertyName*).
- **Variablendeklaration (1)**: Im ersten veränderbaren Codeabschnitt werden die verwendeten Variablen deklariert. Im vorangehenden Codebeispiel ist dies eine einzelne Variable vom Typ *long*.
- **Wertberechnung (2)**: Im nächsten Abschnitt wird die Berechnung des Eigenschaftswertes durchgeführt. Im vorangehenden Codebeispiel wird hier nur der Wert einer WinCC Variablen eingelesen.
- **Wertrückgabe (3)**: Der berechnete Eigenschaftswert ist der Eigenschaft zuzuweisen. Dies erfolgt über die Anweisung *return*.

C-Aktion an einem Ereignis

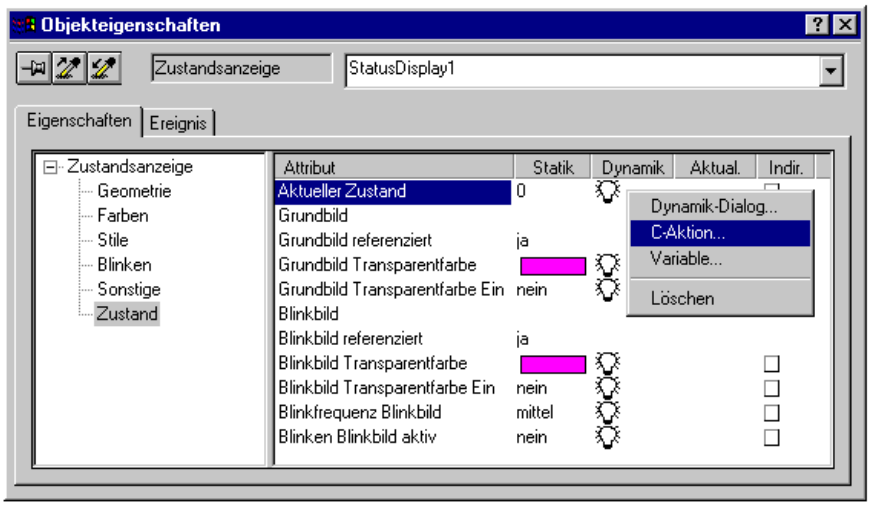
```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{

/*1*/ long lValue;

/*2*/ lValue = GetTagSDWord("S32i_course_test_1");

      SetLeft(lpszPictureName,lpszObjectName,lValue);

}
```

Das vorhergehende Codebeispiel stellt eine typische *C-Aktion* an einem Ereignis dar. Die Bedeutung der einzelnen Abschnitte ist nachfolgend beschrieben.

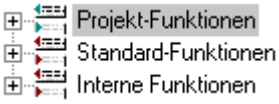
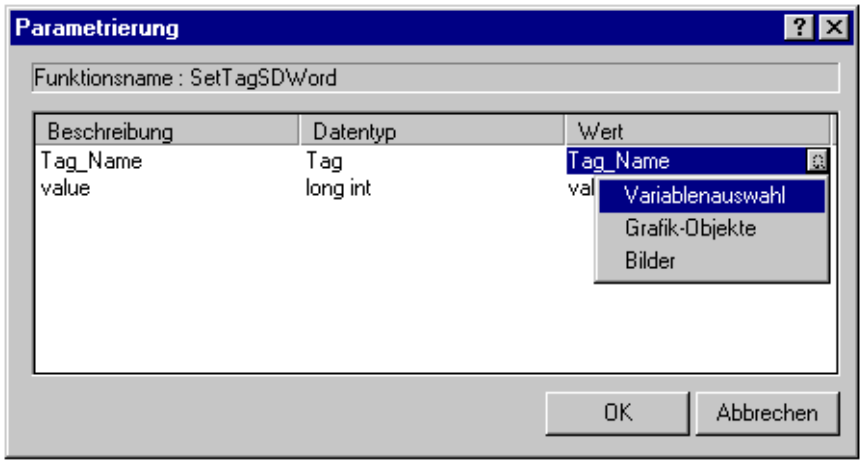
- **Kopf (grau)**: Die ersten drei in der Abbildung grau unterlegten Zeilen bilden den Kopf der *C-Aktion*. Dieser wird automatisch generiert und kann nicht geändert werden. Der Funktionskopf hat für die verschiedenen Arten von Ereignissen ein unterschiedliches Aussehen. Der *C-Aktion* werden die Parameter *lpszPictureName* (Bildname), *lpszObjectName* (Objektname) und *lpszPropertyName* (Eigenschaftsname) übergeben. Der Parameter *lpszPropertyName* enthält dabei jedoch nur bei Ereignissen, die auf die Änderung von Eigenschaften reagieren, relevante Informationen. Zusätzlich können noch weitere ereignisspezifische Parameter übergeben werden.
- **Variablendeklaration (1)**: Im ersten veränderbaren Codeabschnitt werden die verwendeten Variablen deklariert. Im vorangehenden Codebeispiel ist dies eine einzelne Variable vom Typ *long*.
- **Ereignisbehandlung (2)**: Im nächsten Abschnitt werden Aktionen ausgeführt, mit welchen auf das entsprechende Ereignis reagiert werden soll. Im vorangehenden Codebeispiel wird der Wert einer WinCC Variablen eingelesen. Dieser Wert wird dem eigenen Objekt als X Position zugewiesen. Der Rückgabewert einer *C-Aktion* an einem Ereignis ist vom Typ *void*, das heißt es wird kein Rückgabewert erwartet.

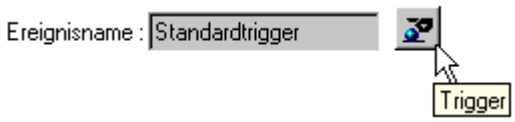
Erstellung einer C-Aktion

In der nachfolgenden Tabelle werden die einzelnen Schritte, welche zur Erstellung einer C-Aktion notwendig sind, beschrieben.

Schritt	Vorgehen: Erstellung einer C-Aktion
1	Öffnen des Editors *Graphics Designer*. Öffnen des gewünschten WinCC Bildes. Öffnen des Dialogs *Objekteigenschaften* des gewünschten Objekts.
2	An der gewünschten Eigenschaft oder dem gewünschten Ereignis ist über **R** und *C-Aktion* der Editor *Aktion editieren* zu öffnen. 
3	Es wird der Editor *Aktion editieren* angezeigt. In diesem ist bereits eine Rahmenfunktion vorhanden. Unter anderem wurde der Kopf der *C-Aktion* automatisch generiert. Dieser kann nicht geändert werden. Im Kopf der *C-Aktion* wird in der ersten Zeile die Datei *apdefap.h* eingebunden. Durch diese Datei werden der *C-Aktion* alle *Projekt-Funktionen*, *Standard-Funktionen* sowie *Internen Funktionen* bekanntgegeben. Der zweite Teil des Kopfes der *C-Aktion* ist der Funktionskopf. Dieser gibt Auskunft über den Rückgabewert der *C-Aktion* sowie übergebene Parameter, welche innerhalb der *C-Aktion* verwendet werden können. Der dritte Teil des Kopfes der *C-Aktion* ist die öffnende geschwungene Klammer. Diese kann nicht gelöscht werden. Zwischen dieser und der ebenso bereits vorhandenen schließenden geschwungenen Klammer ist der eigentliche Code der *C-Aktion* zu programmieren.

Schritt	Vorgehen: Erstellung einer C-Aktion
4	Ein weiterer automatisch generierter Codeabschnitt besteht aus zwei Kommentarblöcken. Diese sind notwendig, um dem Editor *CrossReference* Zugriff auf interne Informationen einer *C-Aktion* zu gewähren. Des weiteren sind sie notwendig, um das Umverdrahten auch innerhalb einer *C-Aktion* zu ermöglichen. Sollen beide Optionen nicht genutzt werden, können diese Kommentare auch problemlos gelöscht werden. Der erste Kommentarblock dient zur Definition der in der *C-Aktion* verwendeten WinCC Variablen. Im Programmcode ist dann anstatt dem wirklichen Variablennamen der definierte Name der Variable zu verwenden. Der zweite Kommentarblock dient zur Definition der in der *C-Aktion* verwendeten WinCC Bilder. Auch hier ist im Programmcode anstatt dem wirklichen Bildnamen der definierte Name des Bildes zu verwenden. Ein Codebeispiel zu diesem Thema finden Sie im Anschluß an diese Tabelle. Es wird beispielhaft die Definition einer WinCC Variable sowie eines WinCC Bildes gezeigt sowie die anschließende Verwendung dieser Definitionen dargestellt.
5	Programmieren des Funktionskörpers, der die gewünschten Berechnungen, Aktionen und dergleichen ausführt. Es stehen verschiedene Hilfsmittel zur Programmierung zur Verfügung. Eines dieser Hilfsmittel ist die Variablenauswahl. Diese ist über den nachfolgend dargestellten Button der Symbolleiste zu öffnen. Im erscheinenden Dialog *Variable* ist eine WinCC Variable auszuwählen und diese Auswahl mit OK zu bestätigen. Der Name der gewählten WinCC Variable wird an der aktuellen Cursorposition in die *C-Aktion* eingefügt. 

Schritt	Vorgehen: Erstellung einer C-Aktion
6	Ein weiteres Hilfsmitte ist die Funktionsauswahl im linken Fenster des Aktionseditors. Über die Funktionsauswahl können alle verfügbaren *Projekt-Funktionen*, *Standard-Funktionen* sowie *Interne Funktionen* automatisch an der aktuellen Cursorposition in die *C-Aktion* eingefügt werden.  Dazu ist die gewünschte Funktion mit **D** auszuwählen. Daraufhin wird ein Dialog *Parametrierung* angezeigt, welcher eine Liste aller zu versorgenden Parameter sowie deren Datentypen enthält. Die Funktion kann in der Spalte *Wert* parametriert werden. Neben der einfachen Texteingabe stehen noch die Optionen *Variablenauswahl*, *Graphik-Objekte* sowie *Bilder* zur Verfügung. Um die Funktion an der aktuellen Cursorposition in die *C-Aktion* einzufügen, ist der Dialog mit *OK* zu bestätigen. 
7	Die fertig programmierte Funktion ist nun zu übersetzen. Dies erfolgt über den nachfolgend dargestellten Button der Symbolleiste.  Das Ergebnis des Übersetzungsvorganges wird in der linken unteren Ecke des Aktionseditors angezeigt. Es wird die Anzahl der aufgetretenen Fehler sowie die Anzahl der Warnungen angezeigt. Fehler führen immer dazu, daß eine *C-Aktion* nicht ausführbar ist. Warnungen sind hingegen Hinweise auf eventuelle Fehler, die bei der Ausführung der *C-Aktion* auftreten könnten. Es gehört zum guten Programmierstil, nur *C-Aktionen* mit dem Ausgabeergebnis *0 Error(s), 0 Warning(s)* zu erstellen. Treten Fehler bei der Übersetzung auf, werden diese im Ausgabefenster angezeigt. Über **D** auf eine Fehlermeldung im Ausgabefenster kann direkt in die entsprechende Codezeile gesprungen werden. 

Schritt	Vorgehen: Erstellung einer C-Aktion
8	Für *C-Aktionen*, welche an einer Eigenschaft eines Objektes erstellt worden sind, ist ein Trigger festzulegen. Bei *C-Aktionen* an Ereignissen ist dies nicht notwendig, da hier das Ereignis selbst den Trigger bildet. Die Festlegung des Triggers erfolgt über den nachfolgend dargestellten Button der Symbolleiste. Es besteht die Möglichkeit der Verwendung von Zeittriggern sowie von Variablentriggern. 
9	Über den Button *OK* des Aktionseditors wird die programmierte *C-Aktion* an die gewünschte Eigenschaft oder das gewünschte Ereignis angefügt. Eine mit einer *C-Aktion* dynamisierte Eigenschaft oder Ereignis wird mit einem grünen Pfeil gekennzeichnet. 

WinCC Variablen und WinCC Bilder definieren

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
// WINCC:TAGNAME_SECTION_START
// syntax: #define TagNameInAction "DMTagName"
#define S32I_COURSE_TEST_1 "S32i_course_test_1"
// next TagID : 1
// WINCC:TAGNAME_SECTION_END

// WINCC:PICNAME_SECTION_START
// syntax: #define PicNameInAction "PictureName"
#define CC_0_STARTPICTURE_00 "cc_0_startpicture_00.Pdl"
// next PicID : 1
// WINCC:PICNAME_SECTION_END

SetTagSDWord(S32I_COURSE_TEST_1,100);

OpenPicture(CC_0_STARTPICTURE_00);

return 0;

}
```

Wird eine neue *C-Aktion* angelegt, enthält diese einen automatisch generierten Codeabschnitt bestehend aus zwei Kommentarblöcken. Diese Kommentarblöcke sind notwendig, um dem Editor *CrossReference* Zugriff auf interne Informationen einer *C-Aktion* zu gewähren. Des weiteren sind sie notwendig, um das Umverdrahten auch innerhalb einer *C-Aktion* zu ermöglichen.

- **Variablendefinition**: Der erste Kommentarblock dient zur Definition der in der *C-Aktion* verwendeten WinCC Variablen. Dieser Kommentarblock beginnt mit der Zeile // *WINCC: TAGNAME_SECTION_START* und endet mit der Zeile // *WINCC: TAGNAME_SECTION_END*. Zwischen diesen beiden Zeilen sind die Namen aller in der *C-Aktion* verwendeten WinCC Variablen zu definieren. Eine Definition erfolgt durch die Präprozessoranweisung *#define* gefolgt vom zu definierenden Namen (im vorangehenden Codebeispiel *S32I_COURSE_TEST_1*) gefolgt vom Namen der WinCC Variablen (im vorangehenden Codebeispiel „*S32i_course_test_1*“).
- **Bilddefinition**: Der zweite Kommentarblock dient zur Definition der in der *C-Aktion* verwendeten WinCC Bilder. Dieser Kommentarblock beginnt mit der Zeile // *WINCC: PICNAME_SECTION_START* und endet mit der Zeile // *WINCC: PICNAME_SECTION_END*. Zwischen diesen beiden Zeilen sind die Namen aller in der *C-Aktion* verwendeten WinCC Bilder zu definieren. Dies erfolgt analog zur zuvor beschriebenen Definition der Variablennamen.
- **Verwendung**: Im eigentlichen Programmcode sind anstatt der wirklichen Variablennamen und Bildnamen die dafür definierten Werte zu verwenden. Der Präprozessor führt vor der Übersetzung der *C-Aktion* einen Textersatz durch, bei dem alle definierten Namen durch die wirklichen Namen ersetzt werden.

4.1.2 Editor Global Script

Der Editor *Global Script* dient zur Erstellung von *Projekt-Funktionen*, *Standard-Funktionen* sowie *Aktionen*.

Projekt-Funktionen

Wird in verschiedenen *C-Aktionen* häufig die gleiche Funktionalität benötigt, kann diese in einer *Projekt-Funktion* formuliert werden. *Projekt-Funktionen* können in allen *C-Aktionen* im WinCC Projekt analog zu allen anderen Funktionen aufgerufen werden. In der nachfolgenden Liste sind die Vorteile der Nutzung von *Projekt-Funktionen* im Gegensatz zur Erstellung des gesamten Programmcodes in *C-Aktionen* aufgeführt.

- **Zentrale Änderbarkeit**: Eine Änderung an der *Projekt-Funktion* wirkt sich auf alle *C-Aktionen* aus, in denen diese verwendet wird. Werden keine *Projekt-Funktionen* verwendet, müssen in diesem Fall alle betroffenen *C-Aktionen* geändert werden. Dies vereinfacht nicht nur die Projektierung, sondern auch die Wartung und die Fehlersuche.
- **Wiederverwendbarkeit**: Wird eine *Projekt-Funktion* einmal programmiert und umfangreich getestet, kann diese jederzeit ohne zusätzlichen Projektierungsaufwand und ohne erneute Tests genutzt werden.
- **Reduziertes Bildvolumen**: Wird nicht der gesamte Programmcode direkt in der *C-Aktion* am Objekt hinterlegt, reduziert dies das Bildvolumen. Damit ergibt sich eine schnellere Bildaufschlagszeit und somit eine höhere Performance im Runtime.
- **Paßwortschutz**: *Projekt-Funktion* können durch Vergabe eines Paßwortes vor Änderungen geschützt werden. Das bedeutet einen Schutz der Projektierungsdaten sowie einen Schutz des eigenen Know-hows.

Projekt-Funktion sind nur projektintern bekannt. Sie werden im Verzeichnis *<WinCC Projektverzeichnis>\LIBRARY* abgelegt und in der Datei *ap_pbib.h* im gleichen Verzeichnis definiert.

Standard-Funktionen

Es stehen bereits eine Vielzahl an *Standard-Funktionen* zur Verfügung. Diese sind im Gegensatz zu den *Projekt-Funktion* in allen WinCC Projekten verfügbar. Bestehende *Standard-Funktionen* können geändert werden. Ebenso können neue *Standard-Funktionen* erstellt werden.

Standard-Funktionen unterscheiden sich von *Projekt-Funktion* nur darin, daß diese projektübergreifend und jene nur projektintern bekannt sind. *Standard-Funktionen* werden im Verzeichnis *<WinCC Installationsverzeichnis>\APLIB* abgelegt und in der Datei *ap_glob.h* im gleichen Verzeichnis definiert.

Interne Funktionen

Neben den *Projekt-Funktion* und den *Standard-Funktionen* gibt es noch die *Internen Funktionen*. Es handelt sich dabei unter anderem um standardmäßige C Funktionen. Diese können vom Anwender nicht geändert werden, es können auch keinen neuen *Internen Funktionen* erstellt werden.

Aktionen

Aktionen werden im Gegensatz zu den zuvor beschriebenen Funktionen nicht aus einer *C-Aktion* oder einer anderen Funktion heraus aufgerufen. Der *Aktion* ist ein Trigger zuzuordnen, welcher ihre Ausführung steuert. Sie wird unabhängig vom aktuell im Runtime angewählten Bild ausgeführt.

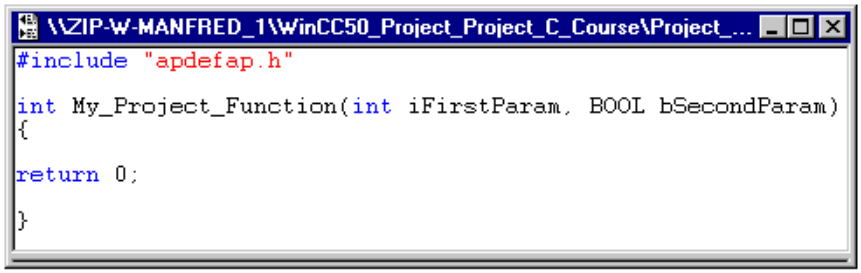
Aktionen können global, das heißt projektweit projektiert werden. In diesem Fall werden sie im Verzeichnis *<WinCC Projektverzeichnis>\PAS* abgelegt. Es können jedoch auch *lokale Aktionen* (rechnerspezifische *Aktionen*) projektiert werden, welche dann im Verzeichnis *<WinCC Projektverzeichnis>\<Rechnername>\PAS* abgelegt werden.

Wird das *Global Script Runtime* in die Anlaufliste des Rechners aufgenommen, werden alle *globalen Aktionen* und alle zum entsprechenden Rechner zugehörigen *lokalen Aktionen* beim Projektstart aktiviert.

Erstellung einer Projekt-Funktion

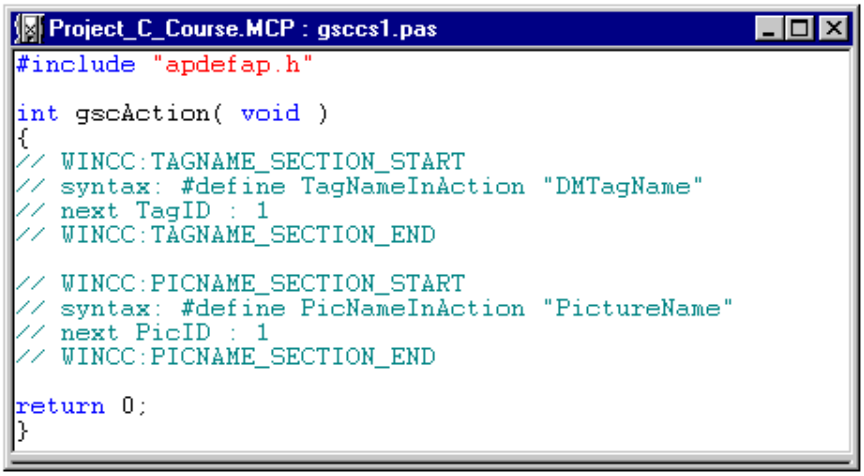
Die Vorgehensweise bei der Erstellung einer *Projekt-Funktion* ist identisch mit der Vorgehensweise bei der Erstellung einer *Standard-Funktionen*. Die folgende Beschreibung gilt also auch für die Erstellung einer *Standard-Funktion*.

Schritt	Vorgehen: Erstellung einer Projekt-Funktion
1	Öffnen des Editors *Global Script*.
2	Über **R** auf dem Punkt *Projekt-Funktionen* und *Neu* → *Funktion* wird der Rahmen einer neuen *Projekt-Funktion* angelegt. 

Schritt	Vorgehen: Erstellung einer Projekt-Funktion
3	Die *Projekt-Funktion* kann vollständig benutzerdefiniert formuliert werden. Es gibt keine nicht veränderbaren Codeteile. Programmieren des Funktionskopfes. Der Funktion ist ein Name zu geben, über welchen sie in *C-Aktionen* oder in anderen Funktionen aufgerufen wird. Des weiteren sind der Rückgabewert sowie die erforderlichen Übergabeparameter der Funktion festzulegen. Sollen andere *Projekt-Funktionen* oder *Standard-Funktionen* in der vorliegenden Funktion verwendet werden, ist die Datei *apdefap.h* einzubinden. Dies erfolgt über die Präprozessoranweisung *#include „apdefap.h“*, welche noch vor dem Funktionskopf einzufügen ist. 
4	Programmieren des Funktionskörpers. Dazu stehen die gleichen Hilfsmittel wie bei der Programmierung von *C-Aktionen* zur Verfügung. Es sind dies die Variablenauswahl sowie die Funktionsauswahl.
5	Die fertig programmierte Funktion ist nun zu übersetzen. Dies erfolgt über den nachfolgend dargestellten Button der Symbolleiste.  Das Ergebnis des Übersetzungsvorganges wird im Ausgabefenster angezeigt. Die aufgetretenen Fehler und Warnungen werden aufgelistet sowie deren Anzahl angezeigt. Über **D** auf eine Fehlermeldung im Ausgabefenster kann direkt in die entsprechende Codezeile gesprungen werden. 
6	Über den nachfolgend dargestellten Button der Symbolleiste kann der *Projekt-Funktion* eine Beschreibung hinzugefügt werden. Im Rahmen dieser Beschreibung kann auch ein Paßwort festgelegt werden, um die *Projekt-Funktion* vor unberechtigtem Zugriff zu schützen. 
7	Die fertig programmierte *Projekt-Funktion* ist unter einem entsprechenden Namen abzuspeichern.

Erstellung einer globalen Aktion

Die Vorgehensweise bei der Erstellung einer *globalen Aktion* ist identisch mit der Vorgehensweise bei der Erstellung einer *lokalen Aktion*. Die folgende Beschreibung gilt also auch für die Erstellung einer *lokalen Aktion*.

Schritt	Vorgehen: Erstellung einer globalen Aktion
1	Öffnen des Editors *Global Script*.
2	Über **R** auf dem Punkt *globale Aktionen* und *Neu* → *Aktion* wird der Rahmen einer neuen *Aktion* angelegt. 
3	Der Kopf der Aktion wurde automatisch generiert und kann nicht geändert werden. Des weiteren sind die zwei Kommentatblöcke zur Definition der WinCC Variablen und WinCC Bilder eingefügt. Die Bedeutung dieser beiden Kommentarblöcke wurde bereits im vorhergehenden Abschnitt bei der Beschreibung der *C-Aktionen* erläutert. 

Schritt	Vorgehen: Erstellung einer globalen Aktion
4	Programmieren des Aktionskörpers. Dazu stehen die gleichen Hilfsmittel wie bei der Programmierung von *C-Aktionen* zur Verfügung. Es sind dies die Variablenauswahl sowie die Funktionsauswahl. Die *Aktion* hat einen Rückgabewert vom Typ *int*. Dieser Rückgabewert kann jedoch vom Anwender nicht ausgewertet werden. Es wird standardmäßig der Wert *0* zurückgegeben.
5	Über den nachfolgend dargestellten Button der Symbolleiste kann der *Aktion* ähnlich wie einer Funktion eine Beschreibung hinzugefügt werden. Es kann hier ebenso ein Paßwort festgelegt werden, um die *Aktion* vor unberechtigtem Zugriff zu schützen. Im Gegensatz zu Funktionen ist jedoch auch ein Trigger einzustellen, welcher die Ausführung der *Aktion* steuert. Bei der Auswahl des Triggers für eine *Aktion* steht dem Anwender eine größere Palette an Optionen zur Verfügung als bei der Auswahl des Triggers einer *C-Aktion* am Objekt. Unter anderem kann auch eine einmalige Ausführung projektiert werden. 
6	Die fertig programmierte *Aktion* ist abzuspeichern.

Testausgabe

Die Ausführung eines Programmes kann durch Testausgaben verfolgt werden. Dies erleichtert während der Entwicklung die Fehlersuche und Fehlerdiagnose. Testausgaben können über die Funktion *printf()* gemacht werden. Mit ihr können einfache Texte, aber auch aktuelle Variablenwerte ausgegeben werden. Um die ausgegebenen Texte sichtbar zu machen, ist ein *Global Script Diagnosefenster* zu projektieren.

Die Funktion printf()

Mit der Funktion *printf()* können Testausgaben gemacht werden. Nachfolgend ist eine beispielhafte Verwendung der Funktion dargestellt.

```
printf("I am %d years old\r\n",iAge);
```

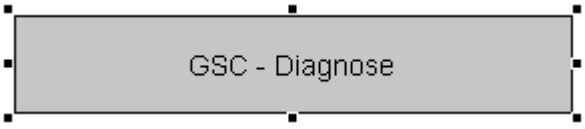
Die Funktion *printf()* verlangt mindestens einen Parameter. Es handelt sich dabei um eine Zeichenkette. Von dieser Zeichenkette hängt die Art und die Anzahl der weiteren zu übergebenen Parameter ab.

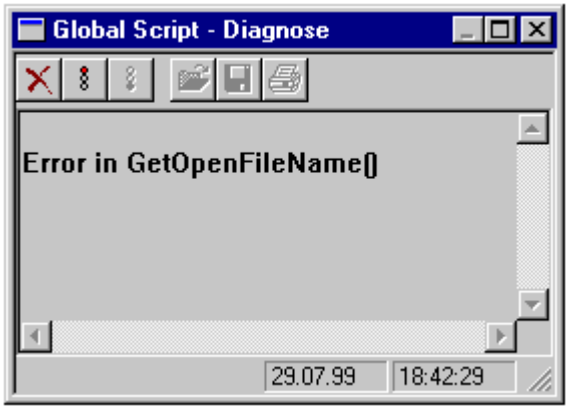
Das Zeichen % dient der Funktion *printf()* als Erkennungszeichen, daß an dieser Position der Wert einer Variable eingefügt werden soll. Das auf das Zeichen % folgende Zeichen bestimmt den Datentyp dieser Variable. Die im vorhergehenden Beispiel verwendete Zeichenkombination *%d* signalisiert die Ausgabe einer Dezimalzahl. Weitere mögliche Kombinationen und deren Bedeutung können der nachfolgenden Tabelle entnommen werden.

Parameter	Bedeutung
%d	Ausgabe einer Dezimalzahl (int oder char)
%ld	Ausgabe einer Variable vom Typ long als Dezimalzahl
%c	Ausgabe eines Zeichens (char)
%x	Ausgabe einer Zahl in Hexadezimalform (mit kleinem a...f)
%X	Ausgabe einer Zahl in Hexadezimalform (mit großem A...F)
%o	Ausgabe einer Zahl in Oktalform
%u	Ausgabe einer Dezimalzahl (nur unsigned Typen)
%f	Ausgabe eines float Wertes in Gleitkommaschreibweise, z.B. 3.43234
%e	Ausgabe einer float Wertes in Exponentialschreibweise, z.B. 23e+432
%E	Wie %e, jedoch mit einem großen E, z.B. 23E+432
%s	Ausgabe einer Zeichenfolge (char*)
%le	Ausgabe eines double Wertes
%%	Ausgabe eines % Zeichens
\n	Ausgabe eines Zeilenwechsels (Carrige Return)
\r	Ausgabe eines Zeilenvorschubs (Line feed)
\t	Ausgabe eines Tabulators
\\	Ausgabe eines \ Zeichens

Global Script Diagnosefenster

Die mit der Funktion *printf()* gemachten Testausgaben werden im *Global Script Diagnosefenster* angezeigt. In der nachfolgenden Tabelle wird die Vorgehensweise zur Projektierung eines solchen *Diagnosefensters* beschrieben.

Schritt	Vorgehen: Global Script Diagnosefenster
1	Öffnen des Editors *Graphics Designer*. Öffnen des gewünschten WinCC Bildes.
2	Ein *Smart-Objekt → Applikationsfenster* projektieren. Nach dessen Plazierung im Bild wird der Dialog *Fensterinhalt* geöffnet. In der Listbox ist der Eintrag *Global Script* auszuwählen. Der Dialog ist mit *OK* zu beenden. Es wird der Dialog *Vorlage* geöffnet. In der Listbox ist der Eintrag *GSC-Diagnose* auszuwählen. Der Dialog ist ebenso mit *OK* zu beenden. 

Schritt	Vorgehen: Global Script Diagnosefenster
3	Um komfortabel mit dem *Global Script Diagnosefenster* arbeiten zu können, ist es ratsam, im *Objekteigenschaftendialog* alle Eigenschaften unter dem Punkt *Sonstige* auf *ja* zu stellen. 
4	Befindet sich das Projekt im Runtime, werden die mit der Funktion *printf()* gemachten Testausgaben im Diagnosefenster angezeigt. Wird die Aktualisierung über den entsprechenden Button der Symbolleiste gestoppt, kann der Fensterinhalt auch gespeichert oder gedruckt werden. 

4.2 Variablen

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema Variablen durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_00.PDL* projektiert.

Variablen

Variablen sind die Datenobjekte, die ein Programm manipuliert. Eine Variable kann erst verwendet werden, nachdem sie definiert worden ist. Alle in einem Programm verwendeten Variablen müssen definiert sein, bevor die erste Anweisung ausgeführt wird.

Variablen kann man mit einem Behälter vergleichen. Mit den Variablennamen geben wir dem Behälter einen eindeutigen Namen. Die Art des Inhalts des Behälters wird durch dessen Datentyp festgelegt. Der anfängliche Inhalt des Behälters wird durch den Initialisierungswert festgelegt. Dieser Inhalt wird meist während der Programmausführung manipuliert.

Die hier beschriebenen Variablen sollten nicht mit WinCC Variablen verwechselt werden. Sie stehen nur innerhalb des Programmcodes zur Verfügung.

Ein Beispiel für die Definition einer Variable zeigt der folgende Programmcode. Dabei wird eine Variable vom Datentyp *int* mit dem Namen *iNumber* definiert. Die Codezeile wird mit einem Semikolon abgeschlossen. Dem Namen der Variable ist ein den Datentyp bezeichnender Präfix vorangestellt. Dies ist nicht zwingend vorgeschrieben, macht jedoch den Datentyp der Variable bei der Programmerstellung sofort ersichtlich.

```
int iNumber;
```

Es besteht auch die Möglichkeit, eine Variable bereits bei deren Definition zu initialisieren.

```
int iNumber = 0;
```

Konstanten

In einem Programm wird außer mit Variablen auch mit Konstanten gearbeitet. Es handelt sich dabei einfach um die direkte Verwendung eines Zahlenwertes. Um die Bedeutung eines solchen Zahlenwertes zu verdeutlichen, kann für diesen mit der Anweisung *#define* eine symbolische Konstante definiert werden.

Ein Beispiel für die Definition einer symbolische Konstante zeigt der folgende Programmcode. Dabei wird die symbolische Konstante *MAX_INT_VALUE* mit dem Zahlenwert *2147483647* definiert. Es ist darauf zu achten, daß die Codezeile nicht mit einem Semikolon abgeschlossen wird. Es ist eine verbreitete Programmierrichtlinie, daß symbolischen Konstanten zur leichteren Unterscheidung von Variablen in Großbuchstaben geschrieben werden.

```
#define MAX_INT_VALUE 2147483647
```

Datentypen

C kennt die in der nachfolgenden Tabelle aufgelisteten Grunddatentypen.

Datentyp	Bedeutung
char	Ein Byte, kann ein Zeichen aufnehmen
int	Ganzzahliger Wert
float	Gleitkommazahl mit einfacher Genauigkeit
double	Gleitkommazahlen mit doppelter Genauigkeit

Eine Variable vom Datentyp *char* hat einen Speicherbedarf von einem Byte. Ihr Inhalt kann sowohl als Zeichen als auch als Zahl interpretiert werden.

Dem Datentyp *int* kann eines der Schlüsselwörter *signed* oder *unsigned* vorangestellt werden. Das Schlüsselwort *signed* bedeutet vorzeichenbehaftet, das Schlüsselwort *unsigned* bedeutet vorzeichenlos.

Dem Datentyp *int* können zusätzlich noch die Schlüsselwörter *long* oder *short* vorangestellt werden. Diese können mit der gleichen Bedeutung auch ohne *int* verwendet werden. Eine Variable vom Datentyp *short* (oder *short int*) hat einen Speicherbedarf von 2 Byte, eine Variable vom Datentyp *long* (oder *long int*) hat ebenso wie eine Variable vom Datentyp *int* einen Speicherbedarf von 4 Byte.

Der Datentyp *double* unterscheidet sich nur durch den Wertebereich vom Datentyp *float* . Dadurch können im Datentyp *double* die Zahlen genauer dargestellt werden. Eine Variable vom Datentyp *float* hat einen Speicherbedarf von 4 Byte, eine Variable vom Datentyp *double* hat einen Speicherbedarf von 8 Byte.

Wertebereiche der Datentypen

Jeder Datentyp ist in der Lage, Zahlenwerte in einem bestimmten Wertebereich darzustellen. Unterschiede ergeben sich aus dem unterschiedlichen Speicherbedarf verschiedener Datentypen sowie dem Unterschied zwischen signed und unsigned Datentypen.

Datentyp	Wertebereich
int	-2 147 483 648 bis 2 147 483 647
unsigned int	0 bis 4 294 967 295
short	-32 768 bis 32 767
unsigned short	0 bis 65 535
long	-2 147 483 648 bis 2 147 483 647
unsigned long	0 bis 4 294 967 295
char	-128 bis 127 (alle ASCII-Zeichen)
unsigned char	0 bis 255 (alle ASCII-Zeichen)
float	-10 ^38 bis 10 ^38
double	-10 ^308 bis 10 ^308

4.2.1 Beispiel 1 – C Datentypen (Ganzzahlen)

In diesem Beispiel werden die in C standardmäßig zur Darstellung von ganzen Zahlen vorhandenen Datentypen verwendet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button1

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

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;
lNumber = -2147483648;
iNumber =  2147483647;

//output in diagnostics window
printf("\r\nExample 1:\r\n");

printf("char:\t\t%d\r\nshort:\t\t%d\r\n"
       "long:\t\t%d\r\nint:\t\t%d\r\n",
       cNumber,sNumber,lNumber,iNumber);

ucNumber = 255;
usNumber = 65535;
ulNumber = 4294967295;
uiNumber = 4294967295;

//output in diagnostics window
printf("unsigned char:\t%u\r\nunsigned short:\t%u\r\n"
       "unsigned long:\t%u\r\nunsigned int:\t%u\r\n",
       ucNumber,usNumber,ulNumber,uiNumber);

}
```

- Bei den ersten drei Zeilen handelt es sich um den Kopf der *C-Aktion*. Dieser kann nicht verändert werden.
- Im nächsten Abschnitt werden die Variablen definiert. Es wird jeweils eine Variable vom Datentyp *char*, *long*, *short* und *int* sowie deren vorzeichenlosen Gegenstücken definiert. Den Namen der Variablen wurde jeweils ein den Datentyp bezeichnender Präfix vorangestellt. Dies ist nicht zwingend vorgeschrieben, macht jedoch den Datentyp einer Variable bei der späteren Programmerstellung sofort ersichtlich. Als Kommentar finden sie in jeder Zeile den von der jeweiligen Variable benötigten Speicherplatz (Kommentare werden mit der Zeichenfolge // eingeleitet und sind grün gekennzeichnet).
- Im nächsten Abschnitt werden den Variablen Werte zugewiesen. Dies erfolgt über den Zuweisungsoperator =. Die im Beispiel verwendeten Zahlenwerte stoßen genau an die Grenzen des darstellbaren Wertebereiches der verschiedenen Datentypen.
- Die Zahlenwerte werden mit der Funktion *printf()* im *Diagnosefenster* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 1:
char:           -128
short:          -32768
long:           -2147483648
int:            2147483647
unsigned char:  255
unsigned short: 65535
unsigned long:  4294967295
unsigned int:   4294967295
```

4.2.2 Beispiel 2 – Definierte Datentypen (Ganzzahlen)

Anstatt der standardmäßig in C vorhandenen Datentypen können auch eigens definierte Datentypen verwendet werden. Bei diesen handelt es sich jedoch nur um Aliasnamen für die eigentlichen Datentypen. In diesem Beispiel werden verschiedene zur Darstellung von ganzen Zahlen definierte Datentypen verwendet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button2

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

CHAR cNumber;           //signed 8 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 = -128;
sNumber = -32768;
lNumber = -2147483648;
iNumber =  2147483647;

//output in diagnostics window
printf("\r\nExample 2:\r\n");

printf("CHAR:\t\t%d\r\nSHORT:\t\t%d\r\n"
       "LONG:\t\t%d\r\nINT:\t\t%d\r\n",
       cNumber,sNumber,lNumber,iNumber);

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

//output in diagnostics window
printf("BOOL:\t\t%u\r\nBYTE:\t\t%u\r\nWORD:\t\t%u\r\n"
       "DWORD:\t\t%u\r\nUINT:\t\t%u\r\n",
       bNumber,byNumber,wNumber,dwNumber,uiNumber);

}
```

- Im ersten Abschnitt werden die Variablen definiert. Es wird jeweils eine Variable vom definierten Datentyp *CHAR*, *SHORT*, *LONG* und *INT* sowie deren vorzeichenlosen Gegenstücken *BYTE*, *WORD*, *DWORD* und *UINT* definiert. Des weiteren wird eine Variable vom Datentyp *BOOL* definiert. Variablen vom Datentyp *BOOL* können die definierten Werte *TRUE* oder *FALSE* zugewiesen werden. Den Namen der Variablen wurde wie im vorangegangenen Beispiel jeweils ein den Datentyp bezeichnender Präfix vorangestellt.
- Im nächsten Abschnitt werden den Variablen Werte zugewiesen. Die im Beispiel verwendeten Zahlenwerte stoßen wiederum genau an die Grenzen des darstellbaren Wertebereichs der verschiedenen Datentypen.
- Die Zahlenwerte werden mit der Funktion *printf()* im *Diagnosefenster* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 2:
CHAR:          -128
SHORT:         -32768
LONG:          -2147483648
INT:           2147483647
BOOL:          1
BYTE:          255
WORD:          65535
DWORD:         4294967295
UINT:          4294967295
```

Typdefinition

Die in diesem Abschnitt verwendeten Datentypen wurden mit der Anweisung *typedef* definiert. Der nachfolgende Programmcode stellt beispielhaft dar, auf welche Weise der Datentyp *BYTE* definiert wurde. *BYTE* steht einfach als Aliasname für den in C standardmäßig vorhandenen Datentyp *unsigned char*. Es können auch eigene Aliasnamen definiert werden.

```
typedef unsigned char BYTE;
```

Der nachfolgenden Tabelle kann entnommen werden, welchem in C standardmäßig vorhandenen Datentyp die einzelnen definierten Datentypen entsprechen.

Definierter Datentyp	C Datentyp
BOOL	int
CHAR	char
SHORT	short
LONG	long
INT	int
BYTE	unsigned char
WORD	unsigned short
DWORD	unsigned long
UINT	unsigned int

4.2.3 Beispiel 3 - WinCC Variablen (Ganzzahlen)

Um mit einer *C-Aktion* oder einer anderen Funktion Aufgaben zur Dynamisierung von Objekten und ähnliches lösen zu können, muß in den meisten Fällen auf WinCC Variablen zugegriffen werden. Zu diesem Zweck sind verschieden Funktionen zum lesen und schreiben des Wertes einer WinCC Variable vorhanden. Diese Funktionen stehen jeweils für jeden in WinCC standardmäßig vorhandenen Variablentyp zur Verfügung. In diesem Beispiel werden verschiedene WinCC Variablen mit Werten beschrieben. Die Inhalte der WinCC Variablen werden über Ausgabefelder angezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button3

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

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 = -128;
sNumber = -32768;
lNumber = -2147483648;

//set wincc tags
SetTagSByte("S08i_course_tag_1",cNumber);
SetTagSWord("S16i_course_tag_1",sNumber);
SetTagSDWord("S32i_course_tag_1",lNumber);

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

//set wincc tags
SetTagBit("BINi_course_tag_1",(SHORT)bNumber);
SetTagByte("U08i_course_tag_1",byNumber);
SetTagWord("U16i_course_tag_1",wNumber);
SetTagDWord("U32i_course_tag_1",dwNumber);

}
```

- Im ersten Abschnitt werden die Variablen definiert. Die Datentypen der Variablen wurden jeweils entsprechend den für WinCC Variablen vorhandenen Datentypen gewählt.
- Im nächsten Abschnitt werden den Variablen Werte zugewiesen. Die im Beispiel verwendeten Zahlenwerte stoßen wiederum genau an die Grenzen des darstellbaren Wertebereichs der verschiedenen Datentypen.
- Die Variablenwerte werden über die entsprechenden Funktionen den verschiedenen WinCC Variablen zugewiesen. Die Funktionsnamen bestehen jeweils aus dem Text *SetTag* sowie einer Bezeichnung des Datentyps der WinCC Variablen, auf welche die jeweilige Funktion anzuwenden ist. Entsprechend den *SetTag* Funktionen zum schreiben von WinCC Variablen sind auch *GetTag* Funktionen zum lesen von WinCC Variablen vorhanden.

- Wird der Funktion *SetTagBit()* eine Variable vom Datentyp *BOOL* (Aliasname für *int*) übergeben, meldet der Compiler eine Warnung. Dies erfolgt, weil die Funktion *SetTagBit()* als Datentyp der übergebenen Variable *SHORT* erwartet. Deshalb wird im vorangehenden Codebeispiel der Inhalt der Variable *bNumber* in *SHORT* umgewandelt, bevor dieser der Funktion *SetTagBit()* übergeben wird. Dieser Vorgang wird auch als *Typecast* (Typumwandlung) bezeichnet.

Typumwandlung

Der Inhalt einer Variable kann, bevor er an eine Funktion übergeben wird oder einer anderen Variable zugewiesen wird, in einen anderen Datentyp umgewandelt werden. Der Datentyp der Variable selbst ändert sich dadurch jedoch nicht. Der nachfolgende Programmcode stellt beispielhaft dar, auf welche Weise der Inhalt einer Variable vom Datentyp *float* in den Datentyp *int* umgewandelt werden kann.

```
iNumber = (int)fNumber;
```

Datentypen von WinCC Variablen

Der nachfolgenden Tabelle kann entnommen werden, welchem in C vorhandenen Datentyp die verschiedenen Datentypen von WinCC Variablen entsprechen. Es handelt sich dabei jeweils um die Datentypen, welche den *SetTag* Funktionen übergeben und welche von den *GetTag* Funktionen zurückgegeben werden.

Datentyp der WinCC Variable	**Datentyp der C Variable**
Vorzeichenbehafteter 8-Bit Wert	char
Vorzeichenbehafteter 16-Bit Wert	short int
Vorzeichenbehafteter 32-Bit Wert	long int
Binäre Variable	short int
Vorzeichenloser 8-Bit Wert	BYTE
Vorzeichenloser 16-Bit Wert	WORD
Vorzeichenloser 32-Bit Wert	DWORD

4.2.4 Beispiel 4 – C Datentypen (Gleitkommazahlen)

In diesem Beispiel werden die in C standardmäßig zur Darstellung von Gleitkommazahlen vorhandenen Datentypen verwendet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button4

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

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:\t%2.17f\tsizeof(float):\t%d\r\n"
       "double:\t%2.17f\tsizeof(double):\t%d\r\n",
       fNumber,sizeof(float),dNumber,sizeof(double));

}
```

- Im ersten Abschnitt werden die Variablen definiert. Es wird jeweils eine Variable vom Datentyp *float* sowie eine Variable vom Datentyp *double* definiert.
- Im nächsten Abschnitt werden den Variablen Werte zugewiesen. Im Beispiel wird beiden Variablen der gleiche Zahlenwert zugewiesen.
- Die Genauigkeit einer Variable vom Typ *float* reicht zirka bis zur siebenten Nachkommastelle. Eine Variable vom Typ *double* kann Zahlen doppelt so genau darstellen. Dies ist durch die Ausgabe der Zahlenwerte mit der Funktion *printf()* im *Diagnosefenster* ersichtlich. Zusätzlich zum Wert der Variablen wird auch noch deren Speicherbedarf ausgegeben. Die Ermittlung des Speicherbedarfs einer Variable oder eines Datentyps erfolgt über die Anweisung *sizeof()*. Der Speicherbedarf wird in Byte angegeben.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 4:
float:    1.00000011920928960    sizeof(float):    4
double:   1.00000010000000010    sizeof(double):   8
```

4.2.5 Beispiel 5 - WinCC Variablen (Gleitkommazahlen)

WinCC Variablen können neben Ganzzahlen auch Gleitkommazahlen enthalten. Dafür stehen zwei den C Datentypen float und double entsprechende Datentypen für WinCC Variablen zur Verfügung. Um auf diese WinCC Variablen lesend oder schreibend zugreifen zu können, sind ebenfalls entsprechende *SetTag* und *GetTag* Funktionen vorhanden. In diesem Beispiel werden verschiedene WinCC Variablen mit Werten beschrieben. Die Inhalte der WinCC Variablen werden über Ausgabefelder angezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button5

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

float fNumber;     //32 bit
double dNumber;    //64 bit

fNumber = 1.0000001;
dNumber = 1.0000001;

//set wincc tags
SetTagFloat("F32i_course_tag_1",fNumber);
SetTagDouble("F64i_course_tag_1",dNumber);

}
```

- Im ersten Abschnitt werden die Variablen definiert. Es wird jeweils eine Variable vom Datentyp *float* sowie eine Variable vom Datentyp *double* definiert.
- Im nächsten Abschnitt werden den Variablen Werte zugewiesen. Im Beispiel wird beiden Variablen wiederum der gleiche Zahlenwert zugewiesen.
- Die Variablenwerte werden über die entsprechenden Funktionen den verschiedenen WinCC Variablen zugewiesen. Entsprechend den hier verwendeten *SetTag* Funktionen zum Schreiben von WinCC Variablen sind auch *GetTag* Funktionen zum Lesen von WinCC Variablen vorhanden.

4.2.6 Beispiel 6 – Statische und externe Variablen

Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis → Maus → Mausklick* projektiert.

Statische Variablen

Eine C Variable ist in einer Funktion ab ihrer Definition gültig. Sie wird bei der Beendigung der Funktion wieder ungültig. Bei einem erneuten Aufruf der Funktion wird sie wieder neu angelegt. Wird einer Variable jedoch das Schlüsselwort *static* vorangestellt, bleibt diese zwischen zwei Funktionsaufrufen erhalten. Sie behält damit auch ihren Wert. Für *C-Aktionen* an Objekten gilt dies jedoch nur solange, wie das WinCC Bild angewählt ist. Wird das Bild abgewählt, werden auch die statische Variable wieder ungültig. Nach erneuter Bildanwahl wird sie bei der ersten Ausführung der *C-Aktion* wieder angelegt.

Externe Variablen

Auf eine C Variable kann nur innerhalb der Funktion zugegriffen werden, in der sie definiert wurde. Wird die Variable jedoch außerhalb jeder Funktion definiert, wird sie zu einer globalen (externen) Variable. Wird diese Variable in einer beliebigen Funktion mit dem Schlüsselwort *extern* deklariert, kann auf sie zugegriffen werden.

Projekt-Funktion CreateExternalTags()

```
int ext_iNumber = 0;

void CreateExternalTags()
{

//nothing to do

}
```

- Die Funktion *CreatExternalTags()* dient nur dem Zweck, eine externe Variable vom Typ *int* zu definieren und zu initialisieren. Die Funktion wird beim Projektstart (am *Ereignis → Sonstige → Bildanwahl* des Startbildes *cc_0_startpicture_00.PDL*) einmal aufgerufen. Ab diesem Zeitpunkt ist die Variable *ext_iNumber* definiert und kann in jeder *C-Aktion* und jeder anderen Funktion verwendet werden.

C-Aktion am Button6

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

//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 opend: %d\r\n",
       ++ext_iNumber,++stat_iNumber);


}
```

- Im ersten Abschnitt wird die externe Variable *ext_iNumber* deklariert, um sie in der *C-Aktion* verwenden zu können.
- Im zweiten Abschnitt wird die statische Variable *stat_iNumber* definiert und initialisiert. Dies wird bei der ersten Ausführung der *C-Aktion* nach der Anwahl des WinCC Bildes ausgeführt. Für die weiteren Ausführungen der *C-Aktion* bleibt der Wert der Variable erhalten. Bei Abwahl des Bildes und erneuter Anwahl wird die Variable neu erzeugt.
- Die Zahlenwerte der Variablen werden mittels des Inkrementoperators ++ um eins erhöht und mit der Funktion *printf()* im *Diagnosefenster* ausgegeben. Somit gibt die Variable ext_iNumber die Anzahl der Klicks auf den Button seit Projektstart an und die Variable stat_iNumber die Anzahl der Klicks auf den Button seit der Anwahl des Bildes an. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 6:
mouseclicks since project was started: 1
mouseclicks since picture was opend: 1
```

4.3 Operatoren und mathematische Funktionen in C

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema Operatoren durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_01.PDL* projektiert.

Operatoren

In einem Programm kontrollieren die Operatoren, was mit Variablen und Konstanten geschehen soll. Variablen und Konstanten werden mit Operatoren verknüpft, um daraus neue Variablenwerte zu erhalten.

Operatoren können in unterschiedliche Kategorien eingeteilt werden. Dazu zählen mathematische Operatoren, bitweise Operatoren und Zuweisungsoperatoren.

Mathematische Operatoren

Operator	Beschreibung
+ (unär)	Positives Vorzeichen (macht eigentlich gar nichts)
- (unär)	Negatives Vorzeichen
+ (binär)	Addition
- (binär)	Subtraktion
*	Multiplikation
/	Division
%	Modulo (gibt den Rest der Division zurück)
++	Inkrementieren
--	Dekrementieren

Bitweise Operatoren

Diese Operatoren ermöglichen es, einzelne Bits in Variablen zu setzen, abzufragen oder rückzusetzen.

Operator	Beschreibung
&	Bitweises UND
|	Bitweises ODER
^	Bitweises exklusives ODER
~	Bitweise Invertierung
<<	Bits verschieben nach links
>>	Bits verschieben nach rechts

Logische Operatoren

Alle logischen Operatoren folgen der Regel, daß 0 FALSE ist und alle anderen Zahlen TRUE sind. Diese Operatoren generieren entweder eine 0 (FALSE) oder eine 1 (TRUE).

Operator	Beschreibung
>	Größer
>=	Größer gleich
==	Gleich
!=	Ungleich
<=	Kleiner gleich
<	Kleiner
&&	Logisches UND
||	Logisches ODER
!	Logische Invertierung

4.3.1 Beispiel 1 - Grundrechnungsarten

In diesem Beispiel werden die grundlegenden mathematischen Operatoren verwendet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button1

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

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("\r\nExample 1\r\n");
printf("%.1f + %.1f = %.1f\r\n",fValue1,fValue2,fResAdd);
printf("%.1f - %.1f = %.1f\r\n",fValue1,fValue2,fResSub);
printf("%.1f * %.1f = %.1f\r\n",fValue1,fValue2,fResMul);
printf("%.1f / %.1f = %.1f\r\n",fValue1,fValue2,fResDiv);

}
```

- Im ersten Abschnitt werden zwei Variablen vom Datentyp *float* definiert und initialisiert. Auf diese zwei Variablen sollen die arithmetischen Operatoren angewandt werden.
- Im nächsten Abschnitt werden weitere vier Variablen vom Datentyp *float* definiert. Diese Variablen sollen die Ergebnisse der durchzuführenden arithmetischen Operationen speichern.
- Im nächsten Abschnitt werden mit den mathematischen Operatoren zum Addieren, Subtrahieren, Multiplizieren und Dividieren Berechnungen durchgeführt.
- Die Ergebnisse der Berechnungen werden mit der Funktion *printf()* im *Diagnosefenster* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
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 Beispiel 2 – Inkrement und Dekrementoperator

In diesem Beispiel wird der Inkrement sowie der Dekrementoperator verwendet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis* → *Maus* → *Mausklick* projektiert.

Präfix und Postfix

Der Inkrement sowie der Dekrementoperator steht in einer Präfix sowie einer Postfixversion zur Verfügung. Beide Versionen führen die gleiche Aktion aus, sie erhöhen beziehungsweise verringern den Wert der Variable, auf die sie angewandt werden, um eins. Der Unterschied liegt jedoch im Rückgabewert. Die Präfixversion inkrementiert oder dekrementiert den Wert der Variable und gibt diesen neuen Wert zurück. Die Postfixversion gibt den ursprünglichen Wert der Variable zurück, erst dann wird die Variable inkrementiert oder dekrementiert.

```
iValue = ++iCount; //präfix
iValue = iCount++; //postfix
```

C-Aktion am Button2

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

static int stat_iPräfix = 0;
static int stat_iPostfix = 0;

printf("\r\nExample 2\r\n");

//execute operators
printf("Prefix operator on first tag:  %d\r\n",++stat_iPräfix);
printf("Postfix operator on second tag: %d\r\n",stat_iPostfix++);

//check values
printf("Value of first tag after execution: %d\r\n",stat_iPräfix);
printf("Value of second tag after execution: %d\r\n",stat_iPostfix);

}
```

- Im ersten Abschnitt werden zwei Variablen vom Datentyp *int* definiert und initialisiert.
- Auf diese zwei Variablen wird der Inkrementoperator in der Präfix sowie der Postfixversion angewandt. Die Rückgabewerte der Operatoren werden mit der Funktion *printf()* im *Diagnosefenster* ausgegeben. Danach werden die Inhalte der Variablen ebenso mit der Funktion *printf()* im *Diagnosefenster* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt. Es wird ersichtlich, daß die Präfixversion des Inkrementoperators den inkrementierten Variablenwert als Rückgabewert liefert, die Postfixversion jedoch den ursprünglichen Wert der Variable. In beiden Fällen wird die Variable jedoch inkrementiert.

```
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 Beispiel 3 - Bitoperationen

In diesem Beispiel werden die grundlegenden bitweisen Operatoren verwendet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

Beschreibung

Im vorliegenden Beispiel werden die bitweisen Operatoren auf den Inhalt zweier WinCC Variablen (*Vorzeichenlose 16-Bit Werte*) angewandt. Das Resultat der Operation wird in einer weiteren WinCC Variable gleichen Datentyps abgelegt. Welcher Operator angewandt werden soll, wird über den Objekt *Button6* gesteuert und gleichzeitig angezeigt. Es stehen die bitweisen Verknüpfungen *AND*, *OR*, *NAND*, *NOR* und *EXOR* zur Auswahl. Jeder dieser Auswahlmöglichkeiten ist ein Zahlenwert zugeordnet, welcher in einer weiteren WinCC Variablen (*Vorzeichenlose 8-Bit Werte*) abgelegt wird.

C-Aktion am Button3

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

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)
{
      //AND
      case 0: dwResult = dwValue1 & dwValue2;
              break;
      //OR
      case 1: dwResult = dwValue1 | dwValue2;
              break;
      //NAND
      case 2: dwResult = ~(dwValue1 & dwValue2);
              break;
      //NOR
      case 3: dwResult = ~(dwValue1 | dwValue2);
              break;
      //EXOR
      case 4: dwResult = dwValue1 ^ dwValue2;
              break;
      //???
      default:return;
}

//write result
SetTagWord("U16i_course_op_3",(WORD)dwResult);

}
```

- Im ersten Abschnitt werden eine Variable vom Datentyp *BYTE* sowie drei Variablen vom Datentyp *DWORD* definiert. Diese Variablen dienen zur Zwischenspeicherung der WinCC Variablen.
- Im nächsten Abschnitt werden die zwei zu verknüpfenden WinCC Variablen in die Variablen *dwValue1* sowie *dwValue2* eingelesen. Des weiteren wird die WinCC Variable, welche die Art der bitweisen Verknüpfungsoperation bestimmt, in die Variable *byOperation* eingelesen.
- Im nächsten Abschnitt werden die beiden Variablen *dwValue1* und *dwValue2* abhängig vom Inhalt der Variable *byOperation* bitweise verknüpft. Das Verknüpfungsergebnis wird in der Variable *dwResult* abgelget. Die Auswahl der auszuführenden Verknüpfungsoperation erfolgt durch eine *switch-case* Konstruktion. Diese wird im Kapitel *Schleifen* näher erläutert.
- Im nächsten Abschnitt wird das in der Variable *dwResult* enthaltene Verknüpfungsergebnis in die entsprechende WinCC Variable geschrieben.

4.3.4 Beispiel 4 – Byteweise Rotieren

In diesem Beispiel werden die bitweisen Schiebeoperatoren dazu verwendet, den in einer WinCC Variable (*Vorzeichenloser 16-Bit Wert*) enthaltenen Wert byteweise zu rotieren. Damit ist eine Vertauschung von Highbyte und Lowbyte gemeint. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button4

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

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);

}
```

- Im ersten Abschnitt wird eine Variable vom Datentyp *DWORD* definiert. Diese Variablen dienen zur Zwischenspeicherung der WinCC Variable. Des weiteren werden zwei Hilfsvariablen vom Typ *DWORD* definiert.
- Im nächsten Abschnitt wird die zu bearbeitende WinCC Variablen in die Variable *dwValue* eingelesen.
- Im nächsten Abschnitt werden die einzelnen Bits der Variable *dwValue* um acht Positionen (ein Byte) nach links geschoben und in der Variable *dwtempValue1* gespeichert. Danach werden die einzelnen Bits der Variable *dwValue* um acht Positionen nach rechts geschoben und in der Variable *dwtempValue2* gespeichert. Die beiden in diesem Abschnitt ermittelten Werte werden bitweise miteinander verknüpft (ODER) und das Ergebnis in der Variable *dwValue* gespeichert.
- Im nächsten Abschnitt wird der in der Variable *dwValue* enthaltene rotierte Variablenwert in die entsprechende WinCC Variable geschrieben.

4.3.5 Beispiel 5 – Mathematische Funktionen

In diesem Beispiel werden verschiedene mathematische Funktionen verwendet, welche standardmäßig in C vorhanden sind. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis* → *Maus* → *Mausklick* projektiert.

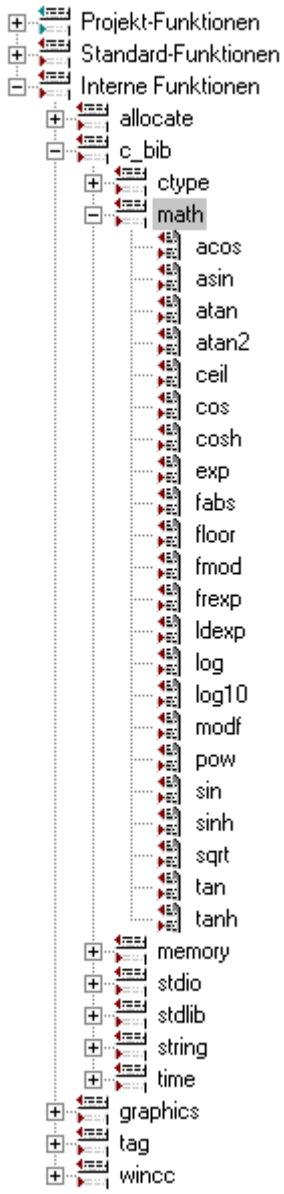
C-Aktion am Button5

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

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("\r\nExample 5\r\n");
printf("%.1f raised to the power of 3 = %.1f\r\n",dValue,dResPow);
printf("Square root of %.1f\t = %.1f\r\n",dValue,dResSqrt);
printf("Absolute value of %d\t = %d\r\n",iValue,iResAbs);
printf("A pseudorandom number\t = %d\r\n",iResRand);

}
```

- Im ersten Abschnitt werden die Variablen definiert.
- Als erstes wird die Funktion *pow()* aufgerufen. Dieser Funktion sind zwei Parameter zu übergeben. Im Beispiel entspricht der Rückgabewert der Funktion der dritten Potenz des Inhalts der Variable *dValue*.
- Als nächstes wird die Funktion *sqrt()* aufgerufen. Diese Funktion liefert als Rückgabewert die Quadratwurzel des übergebenen Wertes.
- Als nächstes wird die Funktion *abs()* aufgerufen. Diese Funktion liefert als Rückgabewert den Absolutwert des übergebenen Wertes.
- Als nächstes wird die Funktion *rand()* aufgerufen. Dieser Funktion sind keine Parameter zu übergeben. Sie liefert als Rückgabewert eine Zufallszahl.
- Die Ergebnisse der Berechnungen werden mit der Funktion *printf()* im *Diagnosefenster* ausgegeben.

Weitere mathematische Funktionen

In der Funktionsauswahl sind die mathematischen Funktionen unter *Interne Funktionen* → *c_bib* → *math* zu finden. Nachfolgender Abbildung können alle vorhandenen mathematischen Funktionen (grau unterlegt) entnommen werden.



4.4 Zeiger

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema Zeiger durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_02.PDL* projektiert.

Arbeiten mit Zeigern

Zeiger sind ein wesentlicher Bestandteil der Sprache C. Ein Zeiger ist eine Variable, die eine Adresse enthält, normalerweise die Adresse einer weiteren Variablen.

Zeiger werden wie normale Variablen definiert. Es wird jedoch dem Namen des Datentyps, auf welchen gezeigt werden soll, das unäre Zeichen * hinzugefügt. Dieses darf nicht mit dem binären Operator * zur Multiplikation zu verwechselt werden. Im nachfolgenden Programmcode wird ein Zeiger auf eine Variable vom Datentyp *int* definiert.

```
int* piValue;
```

Der Inhalt des Zeigers ist nicht definiert. Er zeigt noch auf keine gültige Variable vom Datentyp *int*. Um dies zu verdeutlichen, sollte ein Zeiger bei dessen Definition mit dem Wert *NULL* initialisiert werden. Vor einer Verwendung des Zeigers kann dieser dann auf Gültigkeit geprüft werden.

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

Um den Zeiger auf eine Variable vom Datentyp *int* zeigen zu lassen, ist diesem die Adresse der Variable zuzuweisen. Die erfolgt über den unären Operator &, dem sogenannten Adressoperator. Dieser gibt anstatt dem Wert der Variable dessen Adresse zurück. Im nachfolgenden Programmcode wird dem Zeiger die Adresse einer Variable vom Datentyp *int* zugewiesen.

```
piValue = &iValue;
```

Der Zugriff auf den Wert der Variablen, auf welche gezeigt wird, erfolgt mittels des unären Operators *. Man bezeichnet diesen auch als Inhaltsoperator. Im nachfolgenden Programmcode wird einer Variable vom Datentyp *int* der Wert der Variable, auf welche der Zeiger zeigt, zugewiesen.

```
iValue = *piValue;
```

Arbeiten mit Vektoren

Zeiger und Vektoren hängen sehr eng miteinander zusammen. Im nachfolgenden Programmcode wird ein Vektor bestehend aus 5 Variablen des Datentyps *int* definiert.

```
int iVector[5];
```

Auf die einzelnen Elemente des Vektors kann über deren Index zugegriffen werden. Im nachfolgenden Programmcode wird auf den Inhalt das letzten Elements des Vektors zugegriffen. Dies erfolgt mit dem Indexoperator [].

```
iValue = iVector[4];
```

Der Name des Vektors kann auch als Zeiger auf das erste Element des Vektors verwendet werden. Auf ein bestimmtes Element des Vektors kann auch dadurch zugegriffen werden, daß dieser Zeiger um eine bestimmte Anzahl an Elementen weiterbewegt wird. Dies erfolgt wie im nachfolgenden Programmcode dargestellt durch die Addition eines *int* Wertes zum Zeiger. Auf den Inhalt des resultierenden Zeigers wird über den Inhaltsoperator * zugegriffen. Es wird ebenso wie zuvor auf den letzten Wert des Vektors zugegriffen.

```
iValue = *(iVector+4);
```

Zeichenketten

Eine Zeichenfolge (ein String) kann in C als Vektor aus Zeichen oder als Zeiger auf ein Zeichen definiert werden. Neben den codierten Zeichen fügt C am Ende der Zeichenkette ein Nullzeichen ein. Dieses gilt als Endzeichen der Zeichenkette. Im nachfolgend dargestellten Programmcode wird die Definition von Stringvariablen auf beide Arten dargestellt.

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

Nachfolgend ist die interne Darstellung beider Stringvariablen gezeigt. Im ersten Fall wurde für die Stringvariable genau so viel Speicherplatz reserviert, wie zur Aufnahme des zur Initialisierung angegebenen Strings benötigt wird. Im zweiten Fall wurde so viel Speicherplatz reserviert, wie bei der Definition des Vektors angegeben wurde.

S	t	r	i	n	g	1	\0

S	t	r	i	n	g	2	\0	?	?

4.4.1 Beispiel 1 – Zeiger

In diesem Beispiel werden grundlegende Operationen mit Zeigern durchgeführt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button1

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

int iValue1 = 126;
int iValue2 =  23;

//declare and initialize pointer
int* piValue = NULL;

printf("\r\nExample 1\r\n");
printf("Address: %x\tValue: undefined\r\n",piValue);

//point at iValue1
piValue = &iValue1;
printf("Address: %x\tValue: %d\r\n",piValue,*piValue);

//point at iValue1
piValue = &iValue2;
printf("Address: %x\tValue: %d\r\n",piValue,*piValue);

}
```

- Im ersten Abschnitt werden zwei Variablen vom Datentyp *int* definiert und initialisiert.
- Als nächstes wird ein Zeiger auf eine Variable vom Datentyp *int* definiert und mit *NULL* initialisiert.
- Als nächstes wird die im Zeiger enthaltene Adresse mit der Funktion *printf()* ausgegeben. Der Inhalt, auf den der Zeiger gegenwärtig zeigt, ist nicht definiert. Ein Zugriff auf den Inhalt des Zeigers über den Inhaltsoperator * hätte zu diesem Zeitpunkt eine allgemeine Zugriffsverletzung zur Folge.
- Als nächstes wird dem Zeiger die Adresse der Variable *iValue1* zugewiesen. Deren Adresse sowie deren Inhalt wird wiederum mit der Funktion *printf()* ausgegeben.
- Als nächstes wird dem Zeiger die Adresse der Variable *iValue2* zugewiesen und das Resultat wiederum ausgegeben. Die Ausgabe dieses Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 1
Address: 0          Value: undefined
Address: 2b4f9f8    Value: 126
Address: 2b4f9fc    Value: 23
```

4.4.2 Beispiel 2 – Vektoren

In diesem Beispiel werden grundlegende Operationen mit Vektoren durchgeführt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button2

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

//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\t Value: %d\r\n",iIndex,iValue[iIndex]);
}

}
```

- Im ersten Abschnitt wird ein Vektor bestehend aus 5 Variablen vom Datentyp *int* definiert. Der Vektor wird bei dessen Definition bereits mit Zahlenwerten initialisiert.
- Als nächstes wird eine Zählvariable *iIndex* vom Datentyp *int* definiert.
- Die einzelnen Elemente des Vektors werden mit der Funktion *printf()* ausgegeben. Der Zugriff auf die einzelnen Elemente erfolgt in einer *for* Schleife über den Indexoperator []. Der Umgang mit Schleifen wird im nachfolgenden Kapitel *Schleifen* erläutert. Die Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
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 Beispiel 3 – Zeiger und Vektoren

In diesem Beispiel wird der Zusammenhang zwischen Zeigern und Vektoren erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button3

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

int iValue[] = { 10, 20, 30, 40, 50 };

int iIndex;

int* piElement = NULL;

printf("\r\nExample 3\r\n");

////////////////////////////////////////////////////////////////////
//access via seperat pointer

//point to the first element
piElement = &iValue[0];

printf("Startaddress: %x\r\n",piElement);

for (iIndex = 0; iIndex<5; iIndex++)
{
      printf("Index: %d\t Value: %d\r\n",iIndex,*(piElement+iIndex));
}

printf("\r\n");

////////////////////////////////////////////////////////////////////
//access without seperate pointer
printf("Startaddress: %x\r\n",iValue);

for (iIndex = 0; iIndex<5; iIndex++)
{
      printf("Index: %d\t Value: %d\r\n",iIndex,*(iValue+iIndex));
}

}
```

- Im ersten Abschnitt wird ein Vektor bestehend aus 5 Variablen vom Datentyp *int* definiert. Der Vektor wird bei dessen Definition bereits mit Zahlenwerten initialisiert. In diesem Fall kann die Größenangabe im bei der Definition des Vektors auch weggelassen werden.
- Als nächstes wird eine Zählvariable *iIndex* vom Datentyp *int* definiert.
- Als nächstes wird ein Zeiger *piElement* auf eine Variable vom Datentyp *int* definiert und mit *NULL* initialisiert.
- Als nächstes wird dem Zeiger *piElement* die Adresse des ersten Elements des Vektors zugewiesen. Diese Adresse wird mit der Funktion *printf()* ausgegeben.
- Als nächstes wird über den Zeiger *piElement* auf die einzelnen Elemente des Vektors zugegriffen. Der Zugriff erfolgt in einer *for* Schleife durch Weiterschalten des Zeigers auf die einzelnen Elemente und den Inhaltsoperator *.

- Als nächstes wird wiederum auf die einzelnen Elemente des Vektors zugegriffen. Dieses mal wird jedoch der Name des Vektors selbst als Zeiger verwendet. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
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 Beispiel 4 – Strings

In diesem Beispiel wird das Arbeiten mit Stringvariablen erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button4

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

//declare and initialize string
char szText[13] = "example text";

int i;

printf("\r\nExample 4\r\nCharacters:\r\n");

//access single characters
for (i=0; i<12; i++)
{
      printf("[%c],",szText[i]);
}

printf("\r\n");

//access hole string
printf("String:\r\n%s\r\n",szText);

}
```

- Im ersten Abschnitt wird eine Zeichenfolge (Vektor bestehend aus 13 Zeichen) definiert. Diese Länge ist der Zeichenfolge wurde um ein Zeichen größer als die Länge des zugewiesen Initialisierungsstrings festgelegt, um Platz für das abschließende Nullzeichen zur Verfügung zu stellen.
- Als nächstes wird eine Zählvariable *i* vom Datentyp *int* definiert.
- Als nächstes werden die einzelnen Zeichen der Zeichenfolge mit der Funktion *printf()* ausgegeben. Der Zugriff auf diese Zeichen erfolgt in einer *for* Schleife durch den Indexoperator [].
- Als nächstes wird die gesamte Zeichenfolge mit der Funktion *printf()* ausgegeben. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 4
Characters:
[e],[x],[a],[m],[p],[l],[e],[ ],[t],[e],[x],[t],
String:
example text
```

4.4.5 Beispiel 5 – WinCC Textvariablen

In diesem Beispiel wird der Zusammenhang zwischen Stringvariablen in C und WinCC Textvariablen erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button5

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

//declare and initialize pointer to string
char* pszText = NULL;

//get wincc tag value
pszText = GetTagChar("T08i_course_point_1");

printf("\r\nExample 5\r\n");

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

}
```

- Im ersten Abschnitt wird eine Zeichenfolge (Zeiger auf das erste Zeichen) definiert. Diese wird mit *NULL* initialisiert.
- Als nächstes wird der Inhalt einer WinCC Textvariable über die Funktion *GetTagChar()* eingelesen. Die Funktion reserviert selbst den benötigten Speicherplatz für die Zeichenfolge und gibt deren Anfangsadresse zurück.
- Als nächstes wird die gesamte Zeichenfolge mit der Funktion *printf()* ausgegeben. Des weiteren wird die Länge der Zeichenfolge mit der Funktion *strlen()* ermittelt und ausgegeben sowie die Startadresse der Zeichenfolge ausgegeben. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

Example 5
String: This is an example text!
Stringlength: 24
Startaddress: 1682828

4.5 Schleifen und bedingte Anweisungen

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema Schleifen durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_03.PDL* projektiert.

Schleifen

Mit Hilfe einer Schleife kann ein Codeabschnitt solange ausgeführt werden, solange eine Bedingung wahr ist.

Prinzipiell gibt es zwei Arten von Schleifen: Die vorprüfenden und die nachprüfenden Schleifen. Die vorprüfenden Schleifen prüfen vor dem Schleifenkörper, ob dieser ausgeführt werden soll. Die nachprüfenden Schleifen prüfen nach dem Schleifenkörper, ob dieser ausgeführt werden soll. Daher gilt, daß nachprüfende Schleifen mindestens einmal durchlaufen werden.

Es können die im folgenden aufgeführten Arten von Schleifen unterschieden werden.

while

Nachfolgend ist ein Beispiel für eine *while* Schleife aufgeführt. Die Schleife wird solange wiederholt, solange die Bedingung erfüllt ist. Im vorliegenden Beispiel also solange der Wert der Variable *i* kleiner als 5 ist.

```
int i = 0;

while (i < 5)
{
      //do something
      ++i;
}
```

do-while

Nachfolgend ist ein Beispiel für eine *do-while* Schleife aufgeführt. Die Schleife wird mindestens einmal ausgeführt und dann solange wiederholt, solange die Bedingung erfüllt ist. Im vorliegenden Beispiel also solange der Wert der Variable *i* kleiner als 5 ist.

```
int i = 0;

do
{
      //do something
      ++i;
}
while (i < 5);
```

for

Nachfolgend ist ein Beispiel für eine *for* Schleife aufgeführt. Die Schleife wird solange wiederholt, solange die Bedingung erfüllt ist. Initialisierung des Schleifenzählers sowie die Bearbeitung des Schleifenzählers können in der Schleife formuliert werden.

```
int i = 0;

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

Bedingte Anweisungen

Bei Schleifen wird ein Schleifenkörper bearbeitet, solange eine Bedingung wahr ist. In bedingten Anweisungen wird eine genau einmal bearbeitet, wenn eine Bedingung wahr ist.

Es können die im folgenden aufgeführten Arten von bedingten Anweisungen unterschieden werden.

if-else

Wenn die Bedingung wahr ist, wird die Anweisung im *if* Zweig bearbeitet. Wenn die Bedingung nicht zutrifft, wird die Alternative im *else* Zweig bearbeitet. Der *else* Zweig kann auch weglassen werden, in diesem Fall wird keine Alternative ausgeführt.

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

switch-case

Hier wird eine Variable auf Gleichheit untersucht. Mit *switch* wird die zu untersuchende Variable angegeben. Es wird untersucht, mit welchem *case* Zweig der Wert der Variable übereinstimmt. Dieser *case* Zweig wird dann ausgeführt. Es können beliebig viele *case* Zweige vorhanden sein. Jeder *case* Zweig ist mit einem *break* abzuschließen. Optional kann auch ein *default* Zeig eingefügt werden. Dieser wird dann ausgeführt, wenn der Wert der zu untersuchenden Variable mit keinem *case* Zweig übereinstimmt.

```
switch (i)
{
      case 0: //do something
              break;
      case 1: //do something
              break;
      default://do something default
              break;
}
```

4.5.1 Beispiel 1 – while Schleife

In diesem Beispiel wird die Verwendung der *while* Schleife erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button1

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

//loop count
int iCount = 0;

printf("\r\nExample 1\r\n");

//while loop
while (iCount < 5)
{

      //do something
      printf("Executed loop: iCount = %d\r\n",iCount);

      ++iCount;

}

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


}
```

- Im ersten Abschnitt wird eine Zählvariable *iCount* vom Datentyp *int* definiert und initialisiert.
- Als nächstes wurde eine while Schleife programmiert. Diese wird ausgeführt, solange der Inhalt der Zählvariable *iCount* kleiner als 5 ist. Bei jedem Durchlauf der Schleife wird mit der Funktion *printf()* eine Ausgabe gemacht. Am Ende der Schleife wird die Zählvariable *iCount* um eins erhöht. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
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 Beispiel 2 – do-while Schleife

In diesem Beispiel wird die Verwendung der *do-while* Schleife erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button2

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

//loop count
int iCount = 0;

printf("\r\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);

}
```

- Im ersten Abschnitt wird eine Zählvariable *iCount* vom Datentyp *int* definiert und initialisiert.
- Als nächstes wurde eine do-while Schleife programmiert. Diese wird ausgeführt, solange der Inhalt der Zählvariable *iCount* kleiner als 5 ist. Sie wird jedoch mindestens einmal ausgeführt, da diese Bedingung erst nach dem Durchlauf der Schleife geprüft wird. Bei jedem Durchlauf der Schleife wird mit der Funktion *printf()* eine Ausgabe gemacht. Am Ende der Schleife wird die Zählvariable *iCount* um eins erhöht. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
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 Beispiel 3 – for Schleife

In diesem Beispiel wird die Verwendung der *for* Schleife erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button3

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

//loop count
int iCount = 0;

printf("\r\nExample 3\r\n");

//for loop
for (iCount=0; iCount<5; iCount++)
{

      //do something
      printf("Executed loop: iCount = %d\r\n",iCount);

}

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

}
```

- Im ersten Abschnitt wird eine Zählvariable *iCount* vom Datentyp *int* definiert und initialisiert.
- Als nächstes wurde eine *for* Schleife programmiert. Diese wird ausgeführt, solange der Inhalt der Zählvariable *iCount* kleiner als 5 ist. Die Initialisierung der Zählvariable wird direkt beim Aufruf der Schleife programmiert, ebenso die Aktion zum inkrementierten der Zählvariable. Bei jedem Durchlauf der Schleife wird mit der Funktion *printf()* eine Ausgabe gemacht. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
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 Beispiel 4 – Endlosschleifen

In diesem Beispiel werden Endlosschleifen erläutert. Dies entstehen zumeist unbeabsichtigt durch Programmierfehler, wenn eine Schleifenbedingung immer erfüllt bleibt. Sie können jedoch auch beabsichtigt eingesetzt werden. In diesem Fall muß der Abbruch der Schleife jedoch anders realisiert werden, nämlich die Anweisung *break*. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button4

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

}
```

- Im ersten Abschnitt wird die symbolischen Konstante *MAX_COUNT* definiert. Diese stellt die maximale Ausführungszahl der nachfolgenden Endlosschleife dar.
- Im nächsten Abschnitt wird eine Zählvariable *iCount* vom Datentyp *int* definiert und initialisiert.
- Die aktuelle Anzahl der Schleifendurchläufe soll durch eine Fortschrittsanzeige dargestellte werden. Diese besteht aus einem Balken, dessen Länge die Variable *iProgressBar* enthält, sowie aus einem Statischen Text, dessen Inhalt die Stringvariable *szProgressText* enthält.
- Als nächstes wurde eine Endlosschleife programmiert. Diese könnte auch mit der Anweisung *while(TRUE)* formuliert werden.

- In der Schleife wird die Zählvariable *iCount* überprüft. Falls diese den Wert *MAX_COUNT* überschreitet, wird die Schleife über die Anweisung *break* verlassen.
- Die Zählvariable *iCount* wird inkrementiert.
- Die Fortschrittsanzeige zeigt die bereits durchgeführten Schleifendurchläufe in Prozent an. Bei jedem neu fertiggestellten Prozent werden die Werte der Fortschrittsanzeige neu gesetzt. Wurde kein neues Prozent fertiggestellt, wird sofort ein neuer Schleifendurchlauf mit der Anweisung *continue* begonnen und die restlichen Zeilen übersprungen.
- Die Werte der Fortschrittsanzeige werden gesetzt, indem die Breite des Balkens *ProgressBar* mit der Funktion *SetWidth()* gesetzt wird sowie der Text des Statischen Textes *ProgressText* über die Funktion *SetText()* gesetzt wird. Der verwendete Text wird über die Funktion *sprintf()* zusammengestellt. Diese Funktion arbeitet nach dem Prinzip von *printf()*. Der Text wird jedoch nicht in das *Global Script Diagnosefenster*, sondern in eine Stringvariable geschrieben. Diese ist als erster Parameter der Funktion anzugeben.

4.5.5 Beispiel 5 – if-else Anweisung

In diesem Beispiel wird die Verwendung der *if-else* Anweisung erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button5

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

BYTE byValue;

//get value ti check
byValue = GetTagByte("U08i_course_loop_1");

printf("\r\nExample 5\r\n");

if (byValue < 5)
{

      //do something
      printf("byValue < 5\r\n");

}
else
{

      //do something
      printf("byValue >= 5\r\n");

}

}
```

- Im ersten Abschnitt wird eine Variable *byValue* vom Datentyp *BYTE* definiert. In dieser Variable soll der Inhalt einer WinCC Variable gespeichert werden.
- Im nächsten Abschnitt wird der Inhalt einer WinCC Variable mit der Funktion *GetTagByte()* in die Variable *byValue* eingelesen.
- Als nächstes wurde eine *if-else* Anweisung programmiert. Diese macht abhängig vom Inhalt der Variable *byValue* mit der Funktion *printf()* eine Ausgabe.

4.5.6 Beispiel 6 – switch-case Anweisung

In diesem Beispiel wird die Verwendung der *switch-case* Anweisung erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button6* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button6

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

BYTE byValue;

//get value to check
byValue = GetTagByte("U08i_course_loop_1");

printf("\r\nExample 6\r\n");

switch (byValue)
{
      case 0:     //do something
                  printf("byValue = 0\r\n");
                  break;
      case 1:     //do something
                  printf("byValue = 1\r\n");
                  break;
      case 2:
      case 3:
      case 4:     //do something
                  printf("byValue = 2,3 or 4\r\n");
                  break;
      default:    //do something
                  printf("byValue != 0,1,2,3 and 4\r\n");
                  break;
}

}
```

- Im ersten Abschnitt wird eine Variable *byValue* vom Datentyp *BYTE* definiert. In dieser Variable soll der Inhalt einer WinCC Variable gespeichert werden.
- Im nächsten Abschnitt wird der Inhalt einer WinCC Variable mit der Funktion *GetTagByte()* in die Variable *byValue* eingelesen.
- Als nächstes wurde eine *switch-case* Anweisung programmiert. Diese macht abhängig vom Inhalt der Variable *byValue* mit der Funktion *printf()* eine Ausgabe. Um die gleichen Anweisungen für mehrere verschiedene Zahlenwerte der zu überprüfenden Variable durchzuführen, sind die entsprechenden *case* Zweige untereinander anzuordnen. Die durchzuführenden Anweisungen sind im letzten *case* Zweig zu programmieren.

4.6 Funktionen

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema Funktionen durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_05.PDL* projektiert.

Funktionen

Funktionen dienen dazu, den Programmcode besser strukturieren zu können. Anstatt sich oft wiederholende Anweisungen immer wieder zu programmieren, können diese in eine Funktion verlagert werden. Dadurch wird auch eine zentrale Änderbarkeit und leichtere Wartbarkeit des Programmcodes erreicht.

In WinCC können Funktionen als *Projekt-Funktion* oder als *Standard-Funktionen* erstellt werden.

Übergabeparameter

Funktionen können Werte übergeben werden, abhängig von welchen die Funktion Anweisungen ausführt. Diese Werte können auf verschiedene Arten übergeben werden.

- Es kann ein konstanter Wert übergeben werden.
- Es kann eine Variable übergeben werden. Der Funktion wird jedoch nur der Wert der Variable übergeben. Sie hat keinen Zugriff auch die übergebene Variable selbst.
- Es kann ein Zeiger übergeben werden. Somit erhält die Funktion Zugriff auf die Variable, auf die der Zeiger zeigt. Vektoren und Strukturen können einer Funktion nur über Zeiger übergeben werden.

Rückgabewert

Eine Funktion kann einfach Anweisungen ausführen und nichts zurückgeben. Der Rückgabewert ist dann vom Datentyp *void*. Soll jedoch z.B. eine Berechnung durchgeführt werden, kann der ermittelte Wert über den Rückgabewert an den Aufrufer der Funktion zurückgegeben werden. Hierbei können wiederum Werte oder auch Adressen zurückgegeben werden.

Eine weitere Möglichkeit, Werte an den Aufrufer zurückzugeben, besteht darin, in einen übergebenen Adressbereich zu schreiben. Vektoren oder Strukturen zurückgegeben können nur auf diese Weise zurückgegeben werden

4.6.1 Beispiel 1 – Übergabe von Wertparametern

In diesem Beispiel wird eine einfache Funktion zur Berechnung des Mittelwertes aus drei Zahlen erstellt. Der Funktion werden die Parameter als Werte übergeben, das Ergebnis wird ebenso als Wert zurückgegeben. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis* → *Maus* → *Mausklick* projektiert.

Projekt-Funktion MeanValue()

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

- Im Funktionskopf wird der Name der Funktion mit *MeanValue()* festgelegt. Der Funktion sind drei Variablen vom Datentyp *double* zu übergeben. Zurückgegeben wird ebenso eine Variable vom Datentyp *double*.
- Als nächstes wird eine Variable vom Datentyp *double* definiert, in welcher der Rückgabewert gespeichert werden soll. Dieser wird berechnet, indem die drei übergebenen Werte addiert werden, das Ergebnis dieser Addition wird durch drei dividiert.
- Mit der Anweisung *return* wird das Ergebnis an den Aufrufer der Funktion zurückgegeben.

C-Aktion am Button1

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

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("\r\nExample 1\r\n");

printf("The mean value of %.1f, %.1f and %.1f = %.1f\r\n",
       dValue1,dValue2,dValue3,dMeanValue);

}
```

- Im ersten Abschnitt werden drei Variablen vom Datentyp *double* definiert und initialisiert. Der Mittelwert dieser drei Variablen soll berechnet werden. Eine weitere Variable vom Datentyp *double* wird definiert, um das Ergebnis der Berechnung zu speichern.
- Mit der zuvor erstellten Funktion *MeanValue()* wird der Mittelwert aus den übergebenen Variablen berechnet.
- Das Ergebnis der Berechnung wird mit der Funktion *printf()* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 1
The mean value of 126.2, 23.9 and 45.7 = 65.3
```

4.6.2 Beispiel 2 – Übergabe von Adressparametern

In diesem Beispiel wird eine einfache Funktion zur Berechnung des Mittelwertes der Elemente eines Vektors beliebiger Länge erstellt. Der Funktion wird die Adresse des Vektors sowie dessen Länge übergeben. Das Ergebnis wird als Wert zurückgegeben. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis → Maus → Mausklick* projektiert.

Projekt-Funktion MeanValueVector()

```
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);

}
```

- Im Funktionskopf wird der Name der Funktion mit *MeanValueVector()* festgelegt. Der Funktion ist ein Zeiger auf eine Variablen vom Datentyp *double* zu übergeben. Dieser Zeiger zeigt auf das erste Element des erwarteten Vektors. Des weiteren ist der Funktion die Länge des Vektors zu übergeben. Zurückgegeben wird eine Variable vom Datentyp *double*.
- Als nächstes wird eine Variable vom Datentyp *double* definiert und initialisiert. In dieser soll die Summe der Elemente des übergebenen Vektors gespeichert werden. Diese Summe wird mit Hilfe einer *for* Schleife berechnet.
- Mit der Anweisung *return* wird das Ergebnis an den Aufrufer der Funktion zurückgegeben. Das Ergebnis entspricht der Summe der Elemente des Vektors dividiert durch die Anzahl der Elemente des Vektors.

C-Aktion am Button 2

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

//define  and initialize double vector
double dValue[3] = { 126.2, 23.9, 45.7 };

double dMeanValue;

//calculate mean value of vector
dMeanValue = MeanValueVector(dValue, 3);

//output into diagnostics window
printf("\r\nExample 2\r\n");

printf("The mean value of %.1f, %.1f and %.1f = %.1f\r\n",
       dValue[0],dValue[1],dValue[2],dMeanValue);


}
```

- Im ersten Abschnitt wird ein Vektor bestehend aus drei Variablen vom Datentyp *double* definiert und initialisiert. Der Mittelwert dieser drei Variablen soll berechnet werden. Eine weitere Variable vom Datentyp *double* wird definiert, um das Ergebnis der Berechnung zu speichern.
- Mit der zuvor erstellten Funktion *MeanValueVector()* wird der Mittelwert der Elemente des übergebenen Vektors berechnet.
- Das Ergebnis der Berechnung wird mit der Funktion *printf()* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

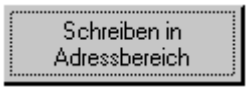
Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 2
The mean value of 126.2, 23.9 and 45.7 = 65.3
```

4.6.3 Schreiben in übergebenen Adressbereich

In diesem Beispiel wird eine einfache Funktion zum Füllen eines Vektors beliebiger Länge mit Zufallszahlen erstellt. Der Funktion wird die Adresse eines Vektors sowie dessen Länge übergeben. Als Rückgabewert zeigt die Funktion über eine Variable vom Typ *BOOL* an, ob die Aktion erfolgreich ausgeführt werden konnte. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.



Projekt-Funktion FillVector()

```
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;

}
```

- Im Funktionskopf wird der Name der Funktion mit *FillVector()* festgelegt. Der Funktion ist ein Zeiger auf eine Variablen vom Datentyp *int* zu übergeben. Dieser Zeiger zeigt auf das erste Element des erwarteten Vektors. Des weiteren ist der Funktion die Länge des Vektors zu übergeben. Zurückgegeben wird eine Variable vom Datentyp *BOOL*, welche anzeigt, ob die Funktion erfolgreich ausgeführt wurde oder nicht.
- Als nächstes wird eine Zählvariable vom Datentyp *int* definiert
- Als nächstes wird der übergebenen Zeiger überprüft. Für die Übergabe der korrekten Länge des Vektors ist der Aufrufer verantwortlich. Wird hier ein falscher Wert übergeben, kann dies zu einer allgemeinen Zugriffsverletzung.
- Mit Hilfe einer *for* Schleife werden die Elemente des übergebenen Vektors über die Funktion *rand()* mit Zufallszahlen gefüllt.

C-Aktion am Button 3

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

#define VECTOR_SIZE 5

//define  int vector
int iVector[VECTOR_SIZE];

int i;

printf("\r\nExample 3\r\n");

//fill vector
if (FillVector(iVector,VECTOR_SIZE) == FALSE)
{
      printf("Error in FillVector\r\n");
      return;
}

printf("Vector Elements: ");

for (i=0; i<VECTOR_SIZE; i++)
{
      printf("[%d] ",iVector[i]);
}

printf("\r\n");

}
```

- Im ersten Abschnitt wird eine symbolische Konstante *VECTOR_SIZE* für die Anzahl der Vektorelemente definiert.
- Als nächstes wird ein Vektor *iVector* bestehend aus *VECTOR_SIZE* Variablen vom Datentyp *int* definiert.
- Als nächstes wird eine Zählvariable *i* vom Datentyp *int* definiert
- Mit der zuvor erstellten Funktion *FillVector()* werden die Elemente des übergebenen Vektors *iVector* mit Zufallszahlen beschrieben. Der Rückgabewert der Funktion *FillVector()* wird direkt bei deren Aufruf wird mit Hilfe einer *if* Anweisung geprüft.
- Die einzelnen Elemente des Vektors *iVector* werden mit der Funktion *printf()* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

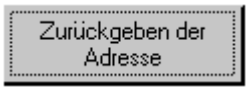
Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

Example 3
Vector Elements: [18467] [6334] [26500] [19169] [15724]

4.6.4 Zurückgeben der Ergebnisadresse

In diesem Beispiel wird eine einfache Funktion erstellt, welche einen Vektor gefüllt mit Zufallszahlen erzeugt. Die Länge des gewünschten Vektors wird der Funktion als Parameter übergeben. Als Rückgabewert liefert die Funktion die Adresse des ersten Elements des erzeugten Vektors. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis → Maus → Mausklick* projektiert.



Projekt-Funktion GetFilledVector()

```
int* GetFilledVector(DWORD dwSize)
{

int* piVector = NULL;

int i;

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

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

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

return piVector;

}
```

- Im Funktionskopf wird der Name der Funktion mit *GetFilledVector()* festgelegt. Der Funktion ist die Anzahl der Elemente des zu erzeugenden Vektors zu übergeben. Zurückgegeben wird der Zeiger auf das erste Element des erzeugten Vektors vom Datentyp *int**.
- Als nächstes wird ein Zeiger *piVector* auf eine Variable vom Datentyp *int* definiert und mit *NULL* initialisiert.
- Als nächstes wird eine Zählvariable *i* vom Datentyp *int* definiert
- Es muß genügend Speicher für den Vektor reserviert werden. Dies erfolgt mit der internen Funktion *SysMalloc()*. Dieser ist die Größe des gewünschten Speicherblocks zu übergeben, welche sich aus dem Speicherbedarf einer Variable vom Datentyp *int* multipliziert mit der gewünschten Anzahl an Verktorelementen berechnet. Die Funktion gibt die Adresse des reservierten Speicherblocks zurück oder *NULL*, wenn nicht genügend Speicher zur Verfügung steht.
- Als nächstes wird die von der Funktion *SysMalloc()* erhaltene Adresse überprüft. Stand nicht genügend Speicher zur Verfügung, wird die Funktion beendet und *NULL* zurückgegeben.
- Mit Hilfe einer *for* Schleife werden die Elemente des Vektors über die Funktion *rand()* mit Zufallszahlen gefüllt.
- Die Adresse des erzeugten Vektors wird über die Anweisung *return* an den Aufrufer zurückgegeben.

C-Aktion am Button4

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

#define VECTOR_SIZE 5

//declare pointer to save address of int vector
int* piVector = NULL;

int i;

printf("\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("\r\n");


}
```

- Im ersten Abschnitt wird eine symbolische Konstante *VECTOR_SIZE* für die Anzahl der Vektorelemente definiert.
- Als nächstes wird ein Zeiger *piVector* auf eine Variable vom Datentyp *int* definiert und mit *NULL* initialisiert.
- Als nächstes wird eine Zählvariable *i* vom Datentyp *int* definiert
- Mit der zuvor erstellten Funktion *GetFilledVector()* wird ein Vektor gefüllt mit Zufallszahlen erzeugt, dessen Adresse im Zeiger *piVector* gespeichert wird. Der Rückgabewert der Funktion *GetFilledVector()* wird danach auf Gültigkeit geprüft.
- Die einzelnen Elemente des erzeugten Vektors werden mit der Funktion *printf()* ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 4
Vector Elements: [11478] [29358] [26962] [24464] [5705]
```

Hinweis:
Wie bei der Übergabe von Strukturen an Funktionen sowie bei der Rückgabe von Strukturen vorzugehen ist, wird im nächsten Kapitel erläutert.

4.7 Strukturen

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema Strukturen durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_04.PDL* projektiert.

Definition eines Strukturtyps

Neben den standardmäßigen Datentypen können mit Hilfe der Strukturen auch eigene Typen definiert werden. Im nachfolgenden Programmcode ist eine prinzipielle Strukturtypdefinition dargestellt. Der definierte Strukturtyp besteht aus einem *int* und einem *float* Stukturelement. Jedem Strukturelement muß ein Name gegeben werden.

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

Verwendung einer Strukturvariable

Nach der Definition des neuen Strukturtyps können Variablen vom Datentyp *struct ExampleStruct* definiert werden. Dies ist im nachfolgenden Programmcode dargestellt. Gleichzeitig wird auch gezeigt, wie auf die Elemente der Strukturvariable zugegriffen wird.

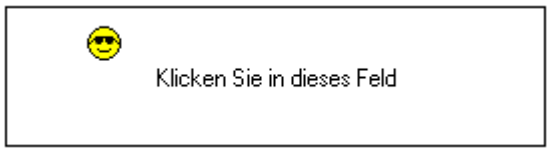
```
struct ExampleStruct esValue;

esValue.iElement = 100;
esValue.fElement = 2.5;
```

Steht anstatt einer Strukturvariable nur ein Zeiger auf eine solche zur Verfügung, kann auf deren einzelnen Elemente wie im nachfolgenden Programmcode dargestellt zugegriffen werden. Dabei ist jedoch darauf zu achten, daß der Zeiger auf eine gültige Strukturvariable zeigt oder zumindest Speicher für diese reserviert wurde.

```
struct ExampleStruct* pesValue;

pesValue->iElement = 100;
pesValue->fElement = 2.5;
```

4.7.1 Beispiel 1 – Strukturvariable

In diesem Beispiel wird die Definition und die Verwendung eines einfachen Strukturtyps erläutert. Das Beispiel ist am nachfolgend dargestellten Objekt *StaticText1* am *Ereignis → Maus → Maus links drücken* projektiert.



C-Aktion am StaticText1

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

//define stucture "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(lpszPictureName,"cool_man",posObject.iLeft);
SetTop(lpszPictureName,"cool_man",posObject.iTop);

}
```

- Im ersten Abschnitt wird ein Strukturtyp *CC_POINT* bestehend aus zwei *int* Elementen definiert. Der Strukturtyp soll zur Aufnahme der Koordinaten eines Mausklicks dienen.
- Als nächstes wird eine Strukturvariable *posObject* vom Datentyp *struct CC_POINT* definiert.
- Als nächstes werden den Elementen der Strukturvariable *posObject* Werte zugewiesen. Bei den zugewiesenen Werten handelt es sich um die Koordinaten des Mausklicks. Diese Koordinaten werden von einer *C-Aktion* am *Ereignis → Maus → Maus links drücken* als Übergabeparameter *x* und *y* geliefert.
- Als nächstes werden die Koordinaten eines Objekts mit den Funktionen *SetLeft()* und *SetTop()* auf die in der Strukturvariable enthaltenen Werte gesetzt.

4.7.2 Beispiel 2 – Typdefinition

In diesem Beispiel wird die Definition und die Verwendung eines einfachen Strukturtyps erläutert. Im Gegensatz zum vorhergehenden Beispiel soll dieser jedoch im gesamten Projekt verfügbar sein und nicht nur in einer *C-Aktion*. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis → Maus → Mausklick* projektiert.

Strukturdefinition in apdefap.h

```
#include "AP_PBIB.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 GFN_SAVE 0
#define GFN_OPEN 1
```

- Es wird der Strukturtyp *tagCC_RECT* bestehend aus vier *int* Elementen definiert. Der Strukturtyp soll zur Aufnahme der Position und Ausmaße eines Rechteckbereiches dienen. Eine Variable dieses Strukturtyps müßte als Variable vom Datentyp *struct tagCC_RECT* definiert werden. Um diese zum Teil umständliche Schreibweise zu vermeiden, wird mit der Anweisung *typedef* ein Aliasname dafür definiert. Soll nun eine Variable dieses Datentyps definiert werden, genügt für den Datentyp die Schreibweise *CC_RECT*. Soll ein Zeiger auf eine Variable dieses Datentyps definiert werden, kann die Schreibweise *PCC_RECT* verwendet werden.

C-Aktion am Button2

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

//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("\r\nExample 2\r\n");

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

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

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

}
```

- Im ersten Abschnitt wird eine Variable *rect* vom Datentyp *CC_RECT* definiert und initialisiert.
- Als nächstes wird eine Variable *prect* vom Datentyp *PCC_RECT* definiert und mit *NULL* initialisiert. Es handelt sich bei diesem Datentyp um einen Zeiger auf eine Variable des Datentyps *CC_RECT*.
- Als nächstes wird auf die Elemente der Strukturvariable *rect* über den Operator . zugegriffen. Deren Inhalt wird über die Funktion *printf()* ausgegeben.
- Als nächstes wird dem Zeiger *prect* die Adresse der Variable *rect* zugewiesen. Als nächstes wird auf die Elemente der Strukturvariable, auf die der Zeiger *prect* zeigt, über den Operator -> zugegriffen. Der Inhalt der Strukturvariable wird wiederum über die Funktion *printf()* ausgegeben. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

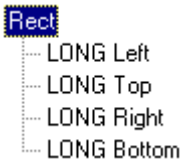
Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 2
Coordinates: 10, 10, 20, 20
Coordinates: 10, 10, 20, 20
```

4.7.3 Beispiel 3 – WinCC Strukturtyp

In diesem Beispiel wird die Definition und die Verwendung eines WinCC Strukturtyps erläutert. Dieser soll im Aufbau dem im vorhergehenden Beispiel definierten Datentyp *CC_RECT* entsprechen. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

WinCC Strukturtyp anlegen

Schritt	Vorgehen: WinCC Strukturtyp anlegen
1	Ein neuer WinCC Strukturtyp wird im *WinCC Explorer* definiert. Über **R** auf dem Punkt *Strukturtypen* und *Neuer Strukturtyp* wird der Dialog zur Definition der Eigenschaften einer neuen WinCC Struktur geöffnet. 
2	Es wird der Dialog *Struktur-Eigenschaften* geöffnet. Der Name des neuen Strukturtyps ist festzulegen. Dies erfolgt über **R** auf dem standardmäßig vergebenen Namen *NeueStruktur* und *Umbenennen*. Als Name wird im Beispiel *Rect* verwendet. 
3	Festlegen der Elemente des neuen Strukturtyps. Ein neues Element wird über den Button *Neues Element* hinzugefügt. Der Name und der Datentyp des neuen Elementes sind über **R** festzulegen. Im Beispiel wird das Element mit *Left* bezeichnet und ist vom Datentyp *LONG*. Für das Element wird die Option *interne Variable* gewählt. Die Namen und Datentypen der restlichen Elemente können der nachfolgenden Abbildung entnommen werden. Der Dialog *Struktur-Eigenschaften* kann mit *OK* geschlossen werden. 

Schritt	Vorgehen: WinCC Strukturtyp anlegen
4	Es können nun WinCC Variablen des Datentyps *Rect* angelegt werden. Dies hat im vorliegenden Beispiel den Effekt, das insgesamt vier Variablen, die den vier Strukturelementen entsprechen, angelegt werden.

Name	Typ
STRi_course_str_1.Right	Vorzeichenbehafteter 32-Bit Wert
STRi_course_str_1.Bottom	Vorzeichenbehafteter 32-Bit Wert
STRi_course_str_1.Left	Vorzeichenbehafteter 32-Bit Wert
STRi_course_str_1.Top	Vorzeichenbehafteter 32-Bit Wert

C-Aktion am Button3

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

//define  CC_RECT structure
CC_RECT rect;

//read element values of wincc structure tag
rect.iLeft = GetTagSDWord("STRi_course_str_1.Left");
rect.iTop = GetTagSDWord("STRi_course_str_1.Top");
rect.iRight = GetTagSDWord("STRi_course_str_1.Right");
rect.iBottom = GetTagSDWord("STRi_course_str_1.Bottom");

printf("\r\nExample 3\r\n");

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

}
```

- Im ersten Abschnitt wird eine Variable *rect* vom Datentyp *CC_RECT* definiert. Der Datentyp *CC_RECT* wurde im vorhergehenden Beispiel definiert.
- Als nächstes werden in die Elemente der Variable *rect* die in einer WinCC Strukturvariable enthaltenen Werte eingelesen. Die in der WinCC Strukturvariablen enthaltenen Werte werden im Beispiel über vier *EA-Felder* angezeigt und können über diese auch verändert werden.
- Als nächstes werden die Elemente der Strukturvariable *rect* mit der Funktion *printf()* ausgegeben. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 3
Coordinates: 0, 0, 0, 0
```

4.7.4 Beispiel 4 – Funktion zum Auslesen eines WinCC Strukturtyps

In diesem Beispiel wird eine Funktion zum Auslesen des im vorhergehenden Beispiel definierten WinCC Strukturtyps erstellt. Diese Funktion kann dann in ähnlicher Weise wie die internen GetTag Funktionen verwendet werden. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis* → *Maus* → *Mausklick* projektiert.

Projekt-Funktion GetTagRect()

```
#include "apdefap.h"
//include file which contains definition of structure tagCC_RECT

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

//allocate memory for CC_RECT structure
prect = SysMalloc(sizeof(CC_RECT));

//check return value of SysMalloc()
if (prect == NULL)
{
      return NULL;
}

//////////////////////////////////////////////////////
//create tag names and get tag values
sprintf(szElementTag,"%s.Left",lpszTagName);
prect->iLeft = GetTagSDWord(szElementTag);

sprintf(szElementTag,"%s.Top",lpszTagName);
prect->iTop = GetTagSDWord(szElementTag);

sprintf(szElementTag,"%s.Right",lpszTagName);
prect->iRight = GetTagSDWord(szElementTag);

sprintf(szElementTag,"%s.Bottom",lpszTagName);
prect->iBottom = GetTagSDWord(szElementTag);
//
//////////////////////////////////////////////////////

//return address of structure as void*
return (void*) prect;

}
```

- Im ersten Abschnitt wird die Datei *apdefap.h* eingebunden, welche die Definition des Strukturtyps *tagCC_RECT* enthält.
- Im Funktionskopf wird der Name der Funktion mit *GetTagRect* festgelegt. Der Funktion ist eine Stringvariable zu übergeben, welche den Namen der auszulesenden WinCC Strukturvariable enthält. Zurückgegeben wird ein Zeiger auf einen Speicherblock nicht näher definierten Datentyps (*void**).
- Im nächsten Abschnitt wird eine Variable *prect* vom Datentyp *PCC_RECT* definiert und mit *NULL* initialisiert. Es handelt sich bei diesem Datentyp um einen Zeiger auf eine Variable des Datentyps *CC_RECT*.
- Als nächstes werden eine Stringvariable zur Aufnahme der Namen der Elemente der WinCC Strukturvariable.

- Als nächstes muß genügend Speicher zur Aufnahme einer Variable vom Datentyp *CC_RECT* reserviert werden. Dies erfolgt mit der internen Funktion *SysMalloc()*. Dieser ist die Größe des gewünschten Speicherblocks, welche mit der Anweisung *sizeof()* ermittelt werden kann, zu übergeben. Die Funktion gibt die Adresse des reservierten Speicherblocks zurück oder *NULL*, wenn nicht genügend Speicher zur Verfügung steht.
- Als nächstes wird die von der Funktion *SysMalloc()* erhaltene Adresse überprüft. Stand nicht genügend Speicher zur Verfügung, wird die Funktion beendet und *NULL* zurückgegeben.
- Als nächstes wird jeweils der Name eines Elements der WinCC Strukturvariable zusammengestellt und der Inhalt des entsprechenden Elements in den reservierten Speicherbereich eingelesen.
- Zurückgegeben wird die Adresse des reservierten Speicherblocks, in welchen der Inhalt der WinCC Strukturvariable gespeichert wurde. Dieser Speicherblock bleibt auch nach dem Verlassen der Funktion reserviert und behält seine Daten.

Hinweis:
Hätte man anstatt der hier vorgestellten Vorgehensweise einfach eine lokale Variable vom Datentyp *CC_RECT* erstellt, deren Elemente mit dem Inhalt der WinCC Strukturvariable beschrieben und deren Adresse zurückgegeben, würde der Aufrufer der Funktion einen ungültigen Zeiger erhalten. Dies liegt darin begründet, daß die Variable vom Datentyp *CC_RECT* am Ende der Funktion ungültig werden würde und somit eine Adresse eines ungültigen Objekts zurückgeliefert werden würde..

C-Aktion am Button4

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

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

//read element values of wincc structure tag
prect = (PCC_RECT) GetTagRect("STRi_course_str_1");

printf("\r\nExample 4\r\n");

//check return value of GetTagRect()
if (prect == NULL)
{
      printf("\r\nError in GetTagRect()\r\n");
      return;
}

//access struct elements
printf("Coordinates: %d, %d, %d, %d\r\n",
        prect->iLeft,prect->iTop,prect->iRight,prect->iBottom);

}
```

- Im ersten Abschnitt wird eine Variable *prect* vom Datentyp *PCC_RECT* definiert und mit *NULL* initialisiert.
- Als nächstes wird mit der zuvor erstellten Funktion *GetTagRect()* eine WinCC Strukturvariable ausgelesen. Die Funktion *GetTagRect()* gibt einen Zeiger auf den Speicherblock zurück, welcher die gewünschten Daten enthält. Dieser Zeiger wird in einen Zeiger vom Typ *PCC_RECT* umgewandelt.
- Als nächstes wird der von der Funktion *GetTagRect()* erhaltene Zeiger überprüft. Steht nicht genügend Speicher zur Verfügung, gibt die Funktion den Wert *NULL* zurück.

- Als nächstes wird jeweils der Name eines Elements der WinCC Strukturvariable zusammengestellt und der Inhalt des entsprechenden Elements in den reservierten Speicherbereich eingelesen.
- Zurückgegeben wird die Adresse des reservierten Speicherblocks, in welchen der Inhalt der WinCC Strukturvariable gespeichert wurde. Dieser Speicherblock bleibt auch nach dem Verlassen der Funktion reserviert und behält seine Daten.
- Als nächstes werden die Elemente der Strukturvariable, auf welche *prect* zeigt, mit der Funktion *printf()* ausgegeben. Die Ausgabe des Programmes ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
Example 4
Coordinates: 0, 0, 0, 0
```

Projekt-Funktion SetTagRect()

Neben der Funktion *GetTagRect()* wurde auch eine entsprechende Funktion *SetTagRect()* erstellt. Diese Funktion wird im Beispiel am *Ereignis → Sonstige → Bildanwahl* des Bildes *cc_9_example_04.PDL* verwendet, um die WinCC Strukturvariable zu initialisieren.

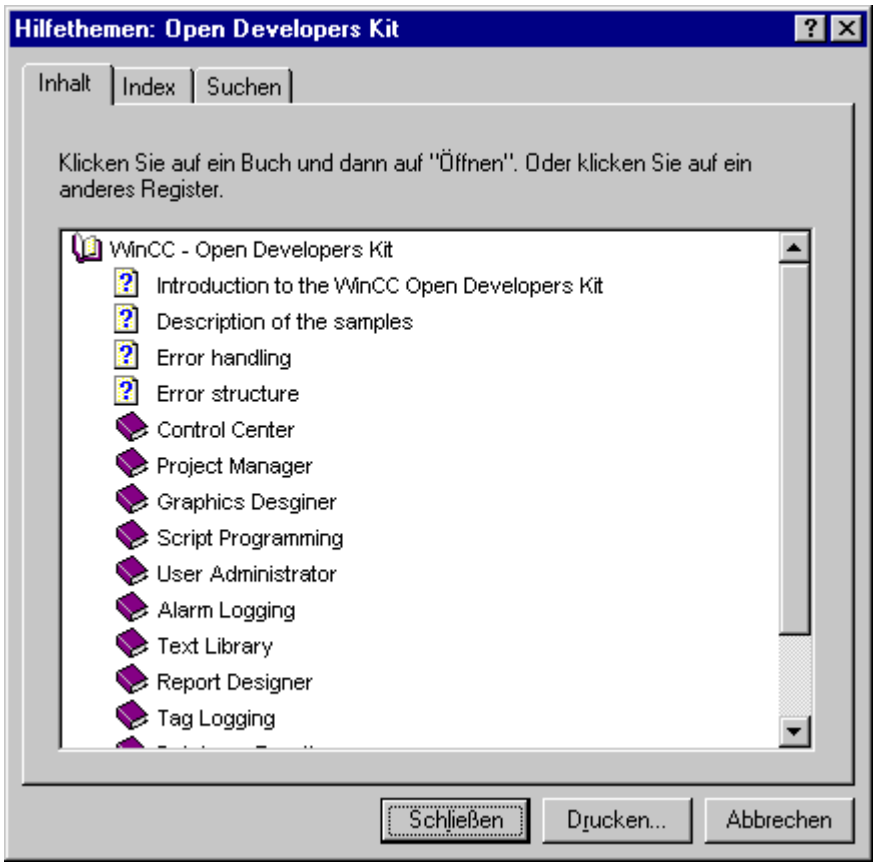
4.8 WinCC API

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema *WinCC API* durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_10.PDL* projektiert.

WinCC Application Program Interface

WinCC stellt als vollkommen offenes und erweiterbares System ein umfangreiches API (Application Program Interface) zur Verfügung. Es handelt sich dabei um eine Schnittstelle, über die Anwendungsprogramme auf WinCC zugreifen können. Die Funktionen des WinCC API können jedoch auch im WinCC Projekt selbst genutzt werden.

Mit dem WinCC ODK (Open Developers Kit) steht eine umfangreiche Beschreibung des WinCC API zur Verfügung. Darin wird das WinCC API anhand von Funktionsbeschreibungen und einer Vielzahl an Beispielen genauestens erläutert. Zusätzlich sind alle Headerdateien mit den benötigten Funktionsdeklarationen enthalten. Das WinCC ODK ist jedoch nicht Bestandteil des WinCC Basispaketes und muß zusätzlich zu diesem erworben werden.



Funktionsbibliotheken

Jede wichtige Applikation von WinCC (*Graphics Designer*, *Tag Logging*, *Alarm Logging* usw.) stellt ein eigenes API zur Verfügung, welches in einer DLL oder auf mehrere DLL's verteilt vorliegt. Unter einer DLL (Dynamic Load Library) versteht man dabei eine dynamisch zu ladende Funktionsbibliothek. Die Deklarationen der in einer DLL vorhandenen Funktionen liegen in einer zugehörigen Headerdatei vor.

Die Einbindung einer DLL in eine *C-Aktion* oder eine andere Funktion wird im nachfolgenden Programmcode beispielhaft gezeigt. In der ersten Zeile wird der Name der DLL angegeben, welche geladen werden soll. Im vorliegenden Beispiel handelt es sich dabei um die DLL, welche die CS Funktionen des *Graphics Designers* enthält. In der zweiten Zeile wird die Headerdatei mit den Funktionsdeklarationen eingebunden. Werden nur eine oder zwei Funktionen benötigt, kann die Deklaration der Funktion auch direkt an dieser Stelle erfolgen. Den Abschluß bildet die Zeile *#pragma code()*. In dem hier gezeigten Beispiel stimmen die Namen von DLL und Headerdatei überein, was auch sinnvoll erscheint. Dies ist leider nicht immer der Fall.

```
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()
```

RT Funktionen und CS Funktionen

Die API Funktionen jeder Applikation kann man grob in zwei unterschiedliche Typen gliedern. Es handelt sich dabei um sogenannte CS Funktionen (Konfigurationssystem) und RT Funktionen (Runtime).

RT Funktionen können im WinCC Projekt zumeist ohne das Laden einer eigenen DLL aufgerufen werden. Die RT Funktionen haben nur Auswirkungen auf das Runtime. Nach dem Neustart des Projektes oder in den meisten Fällen sogar nach einem Bildwechsel gehen Veränderungen, die über RT Funktionen durchgeführt wurden, wieder verloren.

Vor der Verwendung einer CS Funktion im WinCC Projekt muß die entsprechende DLL, in welcher diese programmiert ist, geladen werden. Die Verwendung von CS Funktionen im WinCC Projekt selbst ist jedoch nur in den seltensten Fällen sinnvoll. Im vorliegenden Beispiel wird dies trotzdem gezeigt, da daraus das Grundprinzip der Verwendung der CS Funktionen in eigenen Applikationen abgeleitet werden kann.

Beispielprojekt

Im Beispielprojekt wird keine umfangreiche Erläuterung des API jeder WinCC Applikation durchgeführt. Es werden jedoch die allgemeinen Prinzipien des Umgangs mit dem WinCC API beispielhaft anhand des *Graphics Designers* API vermittelt. Die Beispiele arbeiten mit Objekten im eigens dafür projektierten Bild *cc_9_example_10x.PDL*. Dieses wird über ein *Bildfenster* in dem diesem Kapitel zugeordneten Bild angezeigt.

4.8.1 Beispiel 1 – Eigenschaften ändern über RT Funktion

In diesem Beispiel wird die Verwendung der RT Funktion des *Graphics Designer* API zum Setzen der Eigenschaften von Objekten gezeigt. Es wird die Position eines Objekts durch das Setzen seiner Eigenschaften *PositionX* sowie *PositionY* verändert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button1

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

BOOL bRet = FALSE;

//picture name without extension ".PDL"
char szPictureName[] = "cc_9_example_10ex";
char szObjectName[] = "I/OField1";

//property type
VARTYPE vt = VT_I4;
//new property values
int iValueLeft = 60;
int iValueTop  = 70;

//error structure
CMN_ERROR Error;

////////////////////////////////////////////////////////////
//set the propertys and check the return values
bRet = PDLRTSetPropEx(0,szPictureName,szObjectName,"Left",
            vt,&iValueLeft,NULL,NULL,0,NULL,&Error);
if (bRet == FALSE)
{
      printf("\r\nError in PDLRTSetPropEx()\r\n"
            "\t%s\r\n",Error.szErrorText);
}

bRet = PDLRTSetPropEx(0,szPictureName,szObjectName,"Top",
            vt,&iValueTop,NULL,NULL,0,NULL,&Error);
if (bRet == FALSE)
{
      printf("\r\nError in PDLRTSetPropEx()\r\n"
            "\t%s\r\n",Error.szErrorText);
}
//
////////////////////////////////////////////////////////////

}
```

- Im ersten Abschnitt wird eine Variable *bRet* vom Datentyp *BOOL* definiert und initialisiert. Diese Variable soll den Rückgabewert der aufgerufenen API Funktionen aufnehmen.
- Als nächstes werden zwei Stringvariablen definiert. Über deren Inhalt wird das zu bearbeitende Objekt festgelegt. Es handelt sich dabei um den Bildnamen sowie den Objektnamen. Es ist zu beachten, daß der Bildname nicht die Dateierweiterung *PDL* beinhaltet.
- Zur Festlegung des Typs der zu setzenden Eigenschaften ist eine Variable zu definieren. Dazu steht ein eigener Datentyp zur Verfügung. Es handelt sich dabei um den Datentyp *VARTYPE*. Für jeden vorhandenen Eigenschaftstyp ist eine symbolische Konstante definiert. Die in diesem Beispiel zu setzenden Eigenschaften sind vom Datentyp *VT_I4* (*int* mit einer Länge von 4 Byte).

- Für die zu setzenden Eigenschaften *PositionX* und *PositionY* wird jeweils eine Variable vom Typ *int* definiert, welche den neuen Wert der Eigenschaft enthält.
- Als nächstes wird eine Variable vom Datentyp *CMN_ERROR* definiert. Es handelt sich dabei um eine Struktur, von der beim Fehlschlagen eines Funktionsaufrufes Informationen über den aufgetretenen Fehler erhalten werde können.
- Über die API Funktion *PDLRTSetPropEx()* werden die Eigenschaften *PositionX* und *PositionY* des festgelegten Objekts gesetzt. Der erste Parameter der API Funktion bezeichnet den Adressierungsmodus des Objekts. Die nächsten drei Parameter bezeichnen den gewünschten Bildnamen, Objektnamen sowie Eigenschaftsnamen. Zur Festlegung der gewünschten Eigenschaft ist nicht der deutsche, sondern der englische Name zu verwenden. Im vorhergehenden Beispiel sind dies also *Left* und *Top*. Der nächste Parameter bezeichnet den Eigenschaftstyp. Im darauffolgenden Parameter ist die Adresse der Variable anzugeben, welche den neuen Wert der Eigenschaft enthält. Die nächsten vier Parameter sind für die gewünschte Funktionalität nicht relevant. Im letzten Parameter ist die Adresse der Fehlerstruktur anzugeben.
- Der Rückgabewert jeder API Funktion wird nach deren Aufruf in einer *if* Anweisung geprüft. Schlug der Aufruf fehl, wird dies durch eine Ausgabe angezeigt. Als Teil dieser Ausgabe wird die in der Fehlerstruktur im Strukturelement *szErrorText* erhaltene Information verwendet.

Hinweis:

In diesem Beispiel wird der Rückgabewert der aufgerufenen API Funktion einer Variable vom Datentyp *BOOL* zugewiesen. Diese Variable wird daraufhin geprüft. Der Aufruf der API Funktion und die Überprüfung des Rückgabewertes kann auch in einer Zeile zusammengefaßt werden. Diese Vorgehensweise wird bei den nachfolgenden Beispielen dieses Kapitels verwendet.

4.8.2 Beispiel 2 – Variablenanbindung erstellen über RT Funktion

In diesem Beispiel wird die Verwendung der RT Funktion des *Graphics Designer* API zum Erstellen einer Variablenanbindung gezeigt. An der *Eigenschaft* →*Ausgabe/Eingabe* → *Ausgabewert* eines *EA-Feldes* wird eine Variablenanbindung erstellt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis* →*Maus* →*Mausklick* projektiert.

C-Aktion am Button2

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
//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,"U08i_course_wincc_2");

//set link and check the return value
if ( PDLRTSetLink(0,szPictureName,szObjectName,"OutputValue",
                  &link,NULL,NULL,&Error) == FALSE)
{
      printf("\r\nError in PDLRTSetLink()\r\n%s\r\n",
             Error.szErrorText);
}

}
```

- Im ersten Abschnitt wird die Datei *trigger.h* eingebunden. Diese Datei enthält die Definition einer in diesem Beispiel verwendeten symbolischen Konstante.
- Als nächstes werden zwei Stringvariablen definiert. Über deren Inhalt wird das zu bearbeitende Objekt festgelegt. Es handelt sich dabei um den Bildnamen sowie den Objektnamen.
- Um die Eigenschaften einer Variablenanbindung festlegen zu können, ist ein eigener Datentyp vorhanden. Es handelt sich dabei um den Strukturtyp *LINKINFO*. Es wird eine Variable *link* des Datentyps *LINKINO* definiert.
- Als nächstes wird eine Variable vom Datentyp *CMN_ERROR* definiert.
- Die Strukturelemente der Variable *link* sind mit den Informationen der gewünschten Variablenanbindung zu füllen. Dem Element *LinkType* wird die symbolische Konstante *BUBRT_LT_VARIABLE_DIRECT* zugewiesen. Diese steht für eine direkte Variablenanbindung. Dem Element *dwCycle* wird der Wert *0* zugewiesen, welcher dem Trigger *Bei Änderung* entspricht. Das Element *szLinkName* legt die zu verwendende Variable fest.

- Über die API Funktion *PDLRTSetLink()* wird eine *Variablenanbindung* am festgelegten Objekt erstellt. Der erste Parameter der API Funktion bezeichnet den Adressierungsmodus des Objekts. Die nächsten drei Parameter bezeichnen den gewünschten Bildnamen, Objektnamen sowie Eigenschaftsnamen. Im darauffolgenden Parameter ist die Adresse der Variable *link* anzugeben, welche die zu erstellende Variablenanbindung festlegt. Die nächsten zwei Parameter sind für die gewünschte Funktionalität nicht relevant. Im letzten Parameter ist die Adresse der Fehlerstruktur anzugeben. Schlägt der Aufruf der API Funktion fehl, wird dies durch eine Ausgabe angezeigt.

Hinweis:

Die ersten beiden Beispiele dieses Kapitels beziehen sich auf das im Bild *cc_9_example_10x.PDL* vorhandene Objekt *I/OField1*. Das erste Beispiel verändert die Position des Objekts im Bild, das zweite erstellt an diesem eine Variablenanbindung. Nach einem Bildwechsel gehen die durchgeführten Änderungen wieder verloren. Es ist darauf zu achten, daß nach der Ausführung der im folgenden beschriebenen Beispiele zum Thema CS Funktionen immer ein Bildwechsel durchgeführt wird. Deshalb gehen die mit der Betätigung der ersten beiden Button erreichten Änderungen im Bild nach einer Betätigung eines der restlichen Button wieder verloren.

4.8.3 Beispiel 3 – Neues Objekt erstellen über CS Funktion

In diesem Beispiel wird die Verwendung der CS Funktion des *Graphics Designer* API zum Erstellen eines neuen Objektes gezeigt. Es wird ein neues *EA-Feld* erstellt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button3

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
//PDLCSAPI.dll contains the graphics designer cs functions
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()
//include file with the GUID definitions
#include "pdl_guid.h";

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";//with ".PDL"
DWORD dwFlags = 1;//do not display picture
CMN_ERROR Error;
char szObjectName[] = "I/OField2";
GUID guid = GUID_IOField;//object type i/o field

//get project name
if ( DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE )
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%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%s\r\n",
            Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,dwFlags,&Error) == FALSE )
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
            Error.szErrorText);
      goto OPEN_FAILED;
}
//create object
if ( PDLCSNewObjectEx(szProjectName,szPictureName,&guid,
                        szObjectName,&Error) == FALSE )
{
      printf("\r\nError in PDLCSNewObjectEx()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE )
{
      printf("\r\nError in PDLCSSave()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);
}
```

- Im ersten Abschnitt wird die DLL des *Graphics Designer* API geladen. Zusätzlich wird die Datei *pdl_guid.h* eingebunden, welche die Definition einer in diesem Beispiel verwendeten symbolischen Konstante enthält.
- Als nächstes wird eine Stringvariable *szProjectName* zur Aufnahme des Projektnamens definiert.
- Als nächstes wird eine Stringvariable zur Aufnahme des Bildnamens definiert. Als Besonderheit ist dabei zu beachten, daß der Bildname im Gegensatz zu den RT Funktionen die Dateierweiterung *PDL* beinhalten muß.
- Als nächstes werden die weiteren benötigten Variablen definiert. Unter anderem zählt dazu eine Variable vom Datentyp *GUID*, welche zur Festlegung des zu erstellenden Objekttyps dient.
- Im nächsten Abschnitt wird mit der API Funktion *DMGetRuntimeProject()* der Projektname ermittelt. Der Projektname wird in der Variable *szProjectName* abgelegt. Schlägt die Ermittlung des Projektnamens fehl, wird dies durch eine Ausgabe angezeigt. Die *C-Aktion* wird in diesem Fall mit der Anweisung *return* verlassen.
- Im nächsten Abschnitt wird mit der API Funktion *PDLCSGetOleAppPtr()* das *Graphics Designer* API initialisiert. Schlägt die Initialisierung des *Graphics Designer* API fehl, wird dies durch eine Ausgabe angezeigt. Die *C-Aktion* wird auch in diesem Fall mit der Anweisung *return* verlassen.
- Im nächsten Abschnitt wird mit der API Funktion *PDLCSOpenEx()* das zu bearbeitende Bild geöffnet. Als vorletzten Parameter wird der API Funktion die auf den Wert *1* gesetzte Variable *dwFlags* übergeben. Dies bewirkt, daß das Bild nicht am Bildschirm angezeigt wird. Schlägt das Öffnen des Bildes fehl, wird dies durch eine Ausgabe angezeigt. Über die Anweisung *goto* wird an die Codestelle gesprungen, an der die Verbindung zum *Graphics Designer* API wieder abgebrochen wird.
- Im nächsten Abschnitt wird mit der API Funktion *PDLCSNewObjectEx()* ein neues Objekt mit dem Namen *I/OField2* erstellt. Schlägt die Erstellung des neuen Objektes fehl, wird dies durch eine Ausgabe angezeigt. Über die Anweisung *goto* wird an die Codestelle gesprungen, an der das zuvor geöffnete Bild wieder geschlossen wird.
- Im nächsten Abschnitt wird das Bild mit der API Funktion *PDLCSSave()* abgespeichert. Schlägt das Speichern des Bildes fehl, wird dies durch eine Ausgabe angezeigt. Über die Anweisung *goto* wird ebenso wie im vorhergehenden Abschnitt an die Codestelle gesprungen, an der das zuvor geöffnete Bild wieder geschlossen wird.
- Als nächstes wird mit der *Projekt-Funktion ActualizeObjects()* eine Neuanwahl des bearbeiteten Bildes durchgeführt.
- Als nächstes wird mit der API Funktion *PDLCSClose()* das zuvor geöffnete Bild wieder geschlossen. Vor dieser Anweisung wurde eine Marke eingefügt, welche als Sprungziel für vorhergehende *goto* Anweisungen dient.
- Als nächstes wird mit der API Funktion *PDLCSDelOleAppPtr()* die Verbindung zum *Graphics Designer* API wieder abgebrochen. Vor dieser Anweisung wurde ebenso eine Marke als Sprungziel für eine vorhergehende *goto* Anweisung eingefügt.

Hinweis:

Die *C-Aktionen*, welche für die nachfolgenden Beispiele erstellt wurden, sind der im vorliegenden Beispiel erstellten *C-Aktion* sehr ähnlich. Es wird deshalb bei den nachfolgenden Beispielen auf eine so ausführliche Erläuterung wie im vorliegenden Beispiel verzichtet. Die Codebeschreibungen beschränken sich dort auf eine überblicksmäßige Darstellung des Programmablaufes.

4.8.4 Beispiel 4 – Eigenschaften ändern über CS Funktion

In diesem Beispiel wird die Verwendung der CS Funktion des *Graphics Designer* API zum Setzen der Eigenschaften von Objekten gezeigt. Es wird die Position eines Objekts durch das Setzen seiner Eigenschaften *PostitionX* sowie *PositionY* verändert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button4

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";
char szObjectName[] = "I/OField2";
char szPropertyName[2][5] = { "Left", "Top" };
VARTYPE vt = VT_I4;
int iValue[] = { 60, 130 };
int i;
CMN_ERROR Error;
//get project name
if (DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%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%s\r\n",
             Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,1,&Error) == FALSE)
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
             Error.szErrorText);
      goto OPEN_FAILED;
}
//set propertys
for (i=0; i<2; i++)
{
      if (PDLCSSetPropertyEx(szProjectName,szPictureName,szObjectName,
                  szPropertyName[i],vt,iValue+i,0,NULL,&Error) == FALSE)
      {
            printf("\r\nError in PDLCSSetPropertyEx()\r\n%s\r\n",
            Error.szErrorText);
      }
}
//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE)
{
      printf("\r\nError in PDLCSSave()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);
}
```

- Im ersten Abschnitt wird die DLL des *Graphics Designer* API geladen.
- In zweiten Abschnitt werden die benötigten Variablen definiert. Die Namen der zu setzenden Eigenschaften sowie die zu setzenden Eigenschaftswerte werden im Gegensatz zur Vorgehensweise im Beispiel 1 in Vektoren abgelegt.
- Mit der API Funktion *DMGetRuntimeProject()* wird der Projektname ermittelt.
- Mit der API Funktion *PDLCSGetOleAppPtr()* wird das *Graphics Designer* API initialisiert.
- Mit der API Funktion *PDLCSOpenEx()* wird das zu bearbeitende Bild geöffnet, ohne es jedoch anzuzeigen.
- Innerhalb einer *for* Schleife werden die Eigenschaften des Objektes mit der API Funktion *PDLCSSetPropertyEx()* gesetzt. Sollen Eigenschaften verschiedenen Typs auf diese Art gesetzt werden, ist anstatt der Variable *vt* ebenso ein Vektor zu definieren. Dieser legt dann für jede zu setzende Eigenschaft den Eigenschaftstyp fest.
- Mit der API Funktion *PDLCSSave()* wird das Bild abgespeichert.
- Mit der *Projekt-Funktion ActualizeObjects()* wird eine Neuanwahl des Bildes durchgeführt.
- Mit der API Funktion *PDLCSClose()* wird das Bild wieder geschlossen.
- Mit der API Funktion *PDLCSDelOleAppPtr()* wird die Verbindung zum *Graphics Designer* API wieder abgebrochen.

4.8.5 Beispiel 5 – Variablenanbindung erstellen über CS Funktion

In diesem Beispiel wird die Verwendung der CS Funktion des *Graphics Designer* API zum Erstellen einer Variablenanbindung gezeigt. An der *Eigenschaft* →*Ausgabe/Eingabe* → *Ausgabewert* eines *EA-Feldes* wird eine Variablenanbindung erstellt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis* →*Maus* →*Mausklick* projektiert.

C-Aktion am Button5

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";
char szObjectName[] = "I/OField2";
LINK_INFO link;
CMN_ERROR Error;

//get project name
if (DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%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%s\r\n",
            Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,1,&Error) == FALSE)
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
            Error.szErrorText);
      goto OPEN_FAILED;
}
//set link info struct
link.enumLinkType = BUBRT_LT_VARIABLE_DIRECT;
link.dwCycle = 0;
strcpy(link.szLinkName,"U08i_course_wincc_1");
//set link
if ( PDLCSSetLink(szProjectName,szPictureName,szObjectName,"OutputValue",
            &link,&Error) == FALSE)
{
      printf("\r\nError in PDLCSSetLink()\r\n%s\r\n",
            Error.szErrorText);
      goto ACTION_FAILED;
}
//save picture
if ( PDLCSSave(szProjectName,szPictureName,&Error) == FALSE)
{
      printf("\r\nError in PDLCSSave()\r\n%s\r\n",
      Error.szErrorText);
      goto ACTION_FAILED;
}
//actualize the picture which contains the created object
ActualizeObjects();
//close picture
ACTION_FAILED: PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);
}
```

- Im ersten Abschnitt wird die DLL des *Graphics Designer* API geladen.
- In zweiten Abschnitt werden die benötigten Variablen definiert
- Mit der API Funktion *DMGetRuntimeProject()* wird der Projektname ermittelt.
- Mit der API Funktion *PDLCSGetOleAppPtr()* wird das *Graphics Designer* API initialisiert.
- Mit der API Funktion *PDLCSOpenEx()* wird das zu bearbeitende Bild geöffnet, ohne es jedoch anzuzeigen.
- Im nächsten Abschnitt werden die Strukturelemente der Variable *link* mit den Informationen der gewünschten Variablenanbindung gefüllt.
- Mit der Funktion *PDLCSSetLink()* wird die Variablenanbindung am Objekt erstellt.
- Mit der API Funktion *PDLCSSave()* wird das Bild abgespeichert.
- Mit der *Projekt-Funktion ActualizeObjects()* wird eine Neuanwahl des Bildes durchgeführt.
- Mit der API Funktion *PDLCSClose()* wird das Bild wieder geschlossen.
- Mit der API Funktion *PDLCSDelOleAppPtr()* wird die Verbindung zum *Graphics Designer* API wieder abgebrochen.

4.8.6 Beispiel 6 – Objekte auflisten über CS Funktion

In diesem Beispiel wird die Verwendung der CS Funktion des *Graphics Designer* API zum Auflisten der in einem Bild vorhandenen Objekte gezeigt. Für jedes vorhandene Objekt wird vom API eine eigens dafür zu erstellende Funktion aufgerufen, welcher Informationen über das entsprechende Objekt übergeben werden. Eine solche Funktion wird als Callbackfunktion bezeichnet. Das Beispiel ist am nachfolgend dargestellten Objekt *Button6* am *Ereignis* → *Maus* → *Mausklick* projektiert.

Projektfunktion ObjectCallback()

```
#include "pdlcsapi.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()\r\n");
      return FALSE;
}

//access structure element
printf("%s\r\n",lpInfoStruct->szObjectName);

return TRUE;

}
```

- Im ersten Abschnitt wird die Datei *pdlcsapi.h* eingebunden, welche die Definition des Strukturtyps *OBJECT_INFO_STRUCT* enthält.
- Der Datentyp des Rückgabewertes sowie die Datentypen und die Anzahl der Übergabeparameter ist für diese Funktion festgelegt und kann dem WinCC ODK entnommen werden
- Als nächstes wird ein Zeiger auf eine Variable vom Datentyp *OBJECT_INFO_STRUCT* definiert und initialisiert. Diesem Zeiger wird die im ersten Übergabeparameter erhaltene Adresse zugewiesen. Der Zeiger wird daraufhin auf Gültigkeit geprüft.
- Der Name des Objektes, dessen *OBJECT_INFO_STRUCT* erhalten wurde, wird ausgegeben.

C-Aktion am Button6

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

#pragma code("PDLCSAPI.dll")
#include "pdlcsapi.h"
#pragma code()

char szProjectName[_MAX_PATH];
char szPictureName[] = "cc_9_example_10ex.PDL";
CMN_ERROR Error;

//get project name
if ( DMGetRuntimeProject(szProjectName,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%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%s\r\n",
            Error.szErrorText);
      return;
}
//open picture without displaying it
if ( PDLCSOpenEx(szProjectName,szPictureName,1,&Error) == FALSE)
{
      printf("\r\nError in PDLCSOpenEx()\r\n%s\r\n",
            Error.szErrorText);
      goto OPEN_FAILED;
}
printf("\r\nObjects in picture cc_9_example_10ex.PDL:\r\n");
//enumerate objects
if ( PDLCSEnumObjList(szProjectName,szPictureName,
                      ObjectCallback,NULL,&Error) == FALSE)
{
      printf("\r\nError in PDLCSEnumObjectList()\r\n%s\r\n",
            Error.szErrorText);
}
//close picture
PDLCSClose(szProjectName,szPictureName,&Error);
//disconnect from the API interface of the graphics designer
OPEN_FAILED:   PDLCSDelOleAppPtr(FALSE);

}
```

- Im ersten Abschnitt wird die DLL des *Graphics Designer* API geladen.
- In zweiten Abschnitt werden die benötigten Variablen definiert
- Mit der API Funktion *DMGetRuntimeProject()* wird der Projektname ermittelt.
- Mit der API Funktion *PDLCSGetOleAppPtr()* wird das *Graphics Designer* API initialisiert.
- Mit der API Funktion *PDLCSOpenEx()* wird das zu bearbeitende Bild geöffnet, ohne es jedoch anzuzeigen.
- Mit der API Funktion *PDLCSEnumObjList()* werden alle im Bild vorhandenen Objekte aufgelistet. Dazu ist der API Funktion die Adresse der zuvor erstellten *Projekt-Funktion ObjectCallback()* zu übergeben. Diese Art der Funktion wird auch als Callbackfunktion bezeichnet. Sie wird für jedes im Bild vorhandene Objekt einmal aufgerufen, wobei bei jedem Aufruf jeweils die Daten eines Objekts übergeben werden.
- Mit der API Funktion *PDLCSClose()* wird das Bild wieder geschlossen.
- Mit der API Funktion *PDLCSDelOleAppPtr()* wird die Verbindung zum *Graphics Designer* API wieder abgebrochen.

4.9 Projektumgebung

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema *Projektumgebung* durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_11.PDL* projektiert.

Allgemeines

Es kommt des öfteren vor, daß bei der Programmierung einer *C-Aktion* oder einer anderen Funktion eine Pfadangabe zu einer Datei, der Name des lokalen Rechners und dergleichen anzugeben sind. Diese Werte können dann gemäß den gerade vorliegenden Verhältnissen als absolut vorgegeben eingetragen werden. Damit kommen jedoch Probleme bei der Übertragung eines Projekts auf einen anderen Rechner zustande. Die Verhältnisse dort sind höchstwahrscheinlich anders als auf dem Erstellungssystem eines Projektes. Es ist daher ratsam, bei der Erstellung eines Programmes auf die Verwendung absoluter Pfadangaben und dergleichen zu verzichten. Stattdessen sollten solche Informationen zur Laufzeit ermittelt werden. In diesem Kapitel werden einige Beispiele gezeigt, wie man auf Informationen über die am lokalen Rechner vorliegenden Verhältnisse Zugriff erhält. Dabei wird das WinCC API sowie das Windows API genutzt.

4.9.1 Beispiel 1 – Ermitteln der Projektdatei

In diesem Beispiel wird die Vorgehensweise zur Ermittlung der Projektdatei eines WinCC Projektes gezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button1

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

BOOL bRet;
char szProjectFile[_MAX_PATH+1];
CMN_ERROR Error;

//get the project file *.mcp
bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);

//check return value
if (bRet == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

//display project file
printf("\r\nProjectFile:\r\n%s\r\n",szProjectFile);

}
```

- Im ersten Abschnitt wird eine Variable *bRet* vom Datentyp *BOOL* definiert.
- Als nächstes wird eine Stringvariable *szProjectFile* zur Aufnahme des Projektnamens definiert. Die Länge dieser Variable ist so festzulegen, damit sie auch die längste auftretbare Pfadangabe speichern kann.
- Als nächstes wird eine Variable vom Datentyp *CMN_ERROR* definiert.
- Als nächstes wird mit der API Funktion *DMGetRuntimeProject()* der Projektname ermittelt. Der Projektname wird in der Variable *szProjectFile* abgelegt. Als zweiter Parameter ist die Größe des für den Projektnamen reservierten Speicherplatzes anzugeben. Als dritter Parameter ist die Adresse der Fehlerstruktur anzugeben. Wird keine Fehlerinformation gebraucht, kann hier auch NULL übergeben werden.
- Als nächstes wird der Rückgabewert der API Funktion *DMGetRuntimeProject()* geprüft.
- Als nächstes wird der ermittelte Name der Projektdatei ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
ProjectFile:
\\ZIP-WS5\WinCC50_Project_Project_C_Course\Project_C_Course.MCP
```

4.9.2 Beispiel 2 – Ermitteln des Projektpfades

In diesem Beispiel wird die Vorgehensweise zur Ermittlung des Projektpfades eines WinCC Projektes gezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button2

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

BOOL bRet = FALSE;
char szProjectFile[_MAX_PATH+1];
char* psz = NULL;
CMN_ERROR Error;

//get the project file *.mcp
bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);

//check return value
if (bRet == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%s\r\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("\r\nProjectPath:\r\n%s\r\n",szProjectFile);

}
```

- Im ersten Abschnitt wird eine Variable *bRet* vom Datentyp *BOOL* definiert und initialisiert.
- Als nächstes wird eine Stringvariable *szProjectFile* zur Aufnahme des Projektnamens definiert. Des weiteren wird eine Stringvariable als *char** definiert und mit *NULL* initialisiert.
- Als nächstes wird eine Variable vom Datentyp *CMN_ERROR* definiert.
- Mit der API Funktion *DMGetRuntimeProject()* wird der Projektname ermittelt.
- Als nächstes wird der Rückgabewert der API Funktion *DMGetRuntimeProject()* geprüft.
- Als nächstes wird mit der Funktion *strrchr()* im ermittelten Namen der Projektdatei die Position des letzten \ Zeichens gesucht. Eine Position hinter dem gefundenen Zeichen wird eine 0 eingefügt. Damit bleibt nur die Pfadangabe zur Projektdatei ohne dem Namen der Projektdatei selbst übrig.
- Als nächstes wird der ermittelte Projektpfad ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
ProjectPath:
\\ZIP-WS5\WinCC50_Project_Project_C_Course\
```

4.9.3 Beispiel 3 – Ermitteln des Projektpfades über Projekt-Funktion

In diesem Beispiel wird die im vorhergehenden Beispiel gezeigte Ermittlung des Projektverzeichnisses in eine *Projekt-Funktion* verlagert. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

Projektfunktion GetProjectPath()

```
BOOL GetProjectPath(char* lpstrProjectPath)
{

BOOL bRet = FALSE;
char szProjectFile[_MAX_PATH+1];
char* psz = NULL;
CMN_ERROR Error;

bRet = DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error);

if (bRet == FALSE)
{
      return FALSE;
}

psz = strrchr(szProjectFile,'\\');

if (psz == NULL)
{
      return FALSE;
}

*(psz+1) = 0;

strcpy(lpstrProjectPath,szProjectFile);

return TRUE;

}
```

- Der Projektfunktion ist eine Stringvariable zu übergeben, in welche der ermittelte Projektpfad geschrieben wird. Der Aufrufer der Funktion muß sicherstellen, daß für diese Stringvariable genügend Speicher reserviert wurde. Ob die Funktion erfolgreich ausgeführt wurde, wird durch ihren Rückgabewert angezeigt.
- Bei der Ermittlung des Projektpfades wurde nach dem gleichen Prinzip wie im vorhergehenden Beispiel vorgegangen.
- Der ermittelte Projektpfad ist mit der Funktion *strcpy()* in die übergebene Stringvariable zu kopieren.

C-Aktion am Button3

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

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%s\r\n",szProjectPath);

}
```

- Im ersten Abschnitt wird eine Variable *bRet* vom Datentyp *BOOL* definiert und initialisiert.
- Als nächstes wird eine Stringvariable *szProjectPath* zur Aufnahme des Projektpfades definiert.
- Mit der zuvor erstellten *Projekt-Funktion GetProjectPath()* wird der Projektpfad ermittelt. Danach wird der Rückgabewert der *Projekt-Funktion* geprüft.
- Als nächstes wird der ermittelte Projektpfad ausgegeben.

4.9.4 Beispiel 4 – Ermitteln des Installationsverzeichnisses

In diesem Beispiel wird die Vorgehensweise zur Ermittlung des Installationsverzeichnisses von WinCC gezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button4

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

char szProjectFile[_MAX_PATH+1];
DM_DIRECTORY_INFO dmDirInfo;
CMN_ERROR Error;
char* psz = NULL;

//get the project file *.mcp
if ( DMGetRuntimeProject(szProjectFile,_MAX_PATH+1,&Error) == FALSE)
{
      printf("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

//get wincc directories
if ( DMGetProjectDirectory("",szProjectFile,&dmDirInfo,&Error) == FALSE)
{
      printf("\r\nError in DMGetProjectDirectory()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

if ( (psz = strrchr(dmDirInfo.szGlobalLibDir,'\\')) != NULL)
{
      *psz = 0;
}
if ( (psz = strrchr(dmDirInfo.szGlobalLibDir,'\\')) != NULL)
{
      *(psz+1) = 0;
}

//display installation directory
printf("\r\nInstallationDirectory:\r\n%s\r\n",dmDirInfo.szGlobalLibDir);

}
```

- Im ersten Abschnitt wird eine Stringvariable *szProjectFile* zur Aufnahme des Namens der Projektdatei definiert.
- Als nächstes wird eine Variable *dmDirInfo* zur Aufnahme von Pfadinformationen definiert. Es handelt sich dabei um eine Variable des Strukturtyps *DM_DIRECTORY_INFO*.
- Als nächstes wird eine Variable vom Datentyp *CMN_ERROR* definiert.
- Als nächstes wird eine Stringvariable als *char** definiert und mit *NULL* initialisiert.
- Mit der API Funktion *DMGetRuntimeProject()* wird der Projektname ermittelt.
- Mit der API Funktion *DMGetProjectDirectory()* wird die Variable *dmDirInfo* mit Pfadinformationen gefüllt. Einer der in der Variable erhaltenen Pfade ist der Pfad zum globalen Library Verzeichnis. Dieser Pfad ist im Strukturelement *szGlobalLibDir* gespeichert. Das globale Library Verzeichnis ist ein Unterverzeichnis des WinCC Installationsverzeichnisses.

- Mit der ersten Funktion strrchr() wird das letzte \ Zeichen im ermittelten Pfad gefunden und durch eine 0 ersetzt. Mit der zweiten Funktion strrchr() wird wieder das letzte \ Zeichen im verbliebenen Pfad gesucht. Jetzt wird eine Position dahinter eine 0 eingefügt.
- Als nächstes wird das ermittelte Installationsverzeichnis ausgegeben. Diese Ausgabe ist im nächsten Abschnitt dargestellt.

Ausgabe im Diagnosefenster

Durch das in diesem Abschnitt beschriebene Beispiel wird folgende Ausgabe im *Diagnosefenster* erzeugt.

```
InstallationDirectory:
C:\Siemens\WinCC\
```

4.9.5 Beispiel 5 – Ermitteln des Rechnernamens

In diesem Beispiel wird die Vorgehensweise zur Ermittlung des lokalen Rechnernamens gezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button5

```
#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("\r\nComputerName:\r\nUnknown Computer\r\n");
      return;
}

//display project file
printf("\r\nComputerName:\r\n%s\r\n",szComputerName);

}
```

- Im ersten Abschnitt wird die Windows DLL Kernel32 eingebunden. Da nur eine Funktion aus dieser DLL benötigt wird, wird diese hier direkt deklariert. Zusätzlich wird eine symbolische Konstante für die maximale Länge eines Rechnernamens definiert.
- Als nächstes wird eine Variable *bRet* vom Datentyp *BOOL* definiert und initialisiert.
- Als nächstes wird eine Stringvariable *szComputerName* zur Aufnahme des Rechnernamens definiert. Des weiteren wird eine Variable vom Datentyp *DWORD* definiert, welche mit der Länge der zuvor definierten Stringvariable initialisiert wird.
- Mit der Windows Funktion *GetComputerNameA()* wird der Name des lokalen Rechners ermittelt. Dieser wird in die übergebene Stringvariable *szComputerName* geschrieben.
- Als nächstes wird der Rückgabewert der Windows Funktion *GetComputerNameA()* geprüft.
- Als nächstes wird der ermittelte Rechnername ausgegeben.

4.9.6 Beispiel 6 – Ermitteln des Benutzernamens

In diesem Beispiel wird die Vorgehensweise zur Ermittlung des aktuell in *WindowsNT* angemeldeten Benutzers gezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button6* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button6

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

#pragma code ("advapi32.DLL");
BOOL GetUserNameA(LPSTR UserName, LPDWORD pdwSize);
#define UNLEN 256
#pragma code();

BOOL bRet = FALSE;
char szUserName[UNLEN + 1];
DWORD dwSize = UNLEN + 1;

bRet = GetUserNameA(szUserName,&dwSize);

//check return value
if (bRet == FALSE)
{
      printf("\r\nUserName:\r\nUnknown User\r\n");
      return;
}

//display project file
printf("\r\nUserName:\r\n%s\r\n",szUserName);

}
```

- Im ersten Abschnitt wird die Windows DLL advapi32 eingebunden. Da nur eine Funktion aus dieser DLL benötigt wird, wird diese hier direkt deklariert. Zusätzlich wird eine symbolische Konstante für die maximale Länge eines Benutzernamens definiert.
- Als nächstes wird eine Variable *bRet* vom Datentyp *BOOL* definiert und initialisiert.
- Als nächstes wird eine Stringvariable *szUserName* zur Aufnahme des Benutzernamens definiert. Des weiteren wird eine Variable vom Datentyp *DWORD* definiert, welche mit der Länge der zuvor definierten Stringvariable initialisiert wird.
- Mit der Windows Funktion *GetUserNameA()* wird der Name des aktuell in WindowsNT eingeloggten Benutzers ermittelt. Dieser wird in die übergebene Stringvariable *szUserName* geschrieben.
- Als nächstes wird der Rückgabewert der Windows Funktion *GetComputerNameA()* geprüft.
- Als nächstes wird der ermittelte Benutzername ausgegeben

4.10 Windows API

Die in diesem Kapitel beschriebenen Beispiele sind im WinCC Projekt *Project_C_Course* in den Bildern *cc_0_startpicture_00.PDL* und *cc_2_keyboard_01.PDL* projektiert.

Windows Application Program Interface

Neben dem WinCC API kann in einem WinCC Projekt auch das gesamte Windows API genutzt werden. Damit erhält man nahezu unbeschränkten Zugang zum System.

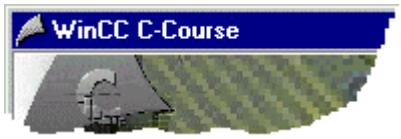
Die folgenden Beispiele vermitteln einen Einblick in dieses Thema. Mit Hilfe dieser Beispiele soll die allgemeine Vorgehensweise bei der Verwendung des Windows API gezeigt werden. Es handelt sich dabei aber nicht um eine erschöpfende Darstellung des Windows API.

Die Funktionen des Windows API liegen genau so wie die Funktionen des WinCC API in verschiedenen DLL's vor. Die Deklarationen der Funktionen liegen in verschiedenen Headerdateien vor. Die Einbindung der DLL's funktioniert nach dem gleichen Prinzip wie die Einbindung der WinCC DLL's. Im nachfolgend dargestellten Programmcode wird beispielhaft die Einbindung

```
#pragma code ("comdlg32.dll")
#include "commdlg.h"
#pragma code()
```

4.10.1 Beispiel 1 – Setzen von Fenstereigenschaften

In diesem Beispiel wird gezeigt, wie die Eigenschaften von Windowsfenstern verändert werden können. Im vorliegenden Beispiel wird der Titel sowie die Geometrie des Runtimefensters geändert. Das Beispiel ist am *Ereignis → Sonstige → Bildanwahl* des Startbildes *cc_0_startpicture_00.PDL* projektiert.



C-Aktion am Startbild

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszI
{

//get handle of runtime window
HWND hWnd = NULL;
hWnd = FindWindow(NULL,"WinCC-Runtime - ");

//set 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);

//set active the first chapter
SetTagByte("U08i_org_bar_1",0);

//
CreateExternalTags();

}
```

- Die in diesem Beispiel verwendeten Windows Funktionen sind im WinCC Projekt bereits bekannt. Daher muß keine Windows DLL mehr geladen werden.
- Im ersten Abschnitt wird eine Variable vom Typ *HWND* definiert und mit *NULL* initialisiert. Es handelt sich bei dieser Variable um ein sogenanntes Fensterhandle. Man kann sich darunter einen Zeiger auf ein Windows Fenster verstehen.
- Mit der Windows Funktion *FindWindow()* kann das Fensterhandle eines Windows Fensters durch die Angabe seines Fenstertitels ermittelt werden. Wird der standardmäßige Titel des Runtimefensters angegeben, kann dessen Fensterhandle ermittelt werden.
- Mit der Windows Funktion *SetWindowText()* kann der Titel des Runtimefensters verändert werden. Im vorliegenden Beispiel wird dieser auf *WinCC C-Course* geändert.
- Mit der Windows Funktion *SetWindowPos()* können die Position und die Größe des Runtimefensters am Bildschirm festgelegt werden. Im vorliegenden Beispiel wird das Runtimefenster in die linke obere Ecke (Position 0/0) des Bildschirms gesetzt, die Größe des Runtimefensters wird auf 1024 mal 768 gesetzt.
- Die restlichen Anweisungen im oben dargestellten Progammcode führen Initialisierungen durch, die für dieses Beispiel nicht relevant sind.

4.10.2 Beispiel 2 – Auslesen der Systemzeit

In diesem Beispiel wird gezeigt, wie die Systemzeit ausgelesen und angezeigt werden kann. Im vorliegenden Beispiel wird die Zeit sowie das Datum angezeigt. Das Beispiel ist im Bild *cc_0_startpicture_00.PDL* projektiert.

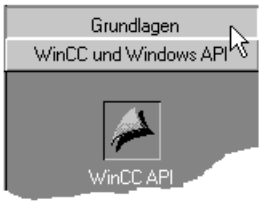
C-Aktion am Statischen Text Time

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

#pragma code("kernel32.dll")
VOID GetLocalTime(LPSYSTEMTIME lpSystemTime);
#pragma code()

SYSTEMTIME sysTime;
char szTime[6] = "";

GetLocalTime(&sysTime);

sprintf(szTime,"%02d:%02d",sysTime.wHour,sysTime.wMinute);

return szTime;

}
```

- Im ersten Abschnitt wird die Windows DLL kernel32 eingebunden. Da nur eine Funktion aus dieser DLL benötigt wird, wird diese hier direkt deklariert.
- Als nächstes wird eine Variable *sysTime* vom Datentyp *SYSTEMTIME* definiert. Es handelt sich dabei um einen Strukturtyp, welcher zum Speichern der Systemzeit dient.
- Als nächstes wird eine Stringvariable *szTime* zur Aufnahme der aktuellen Zeit im Format hh:mm.
- Mit der Windows Funktion *GetLocalTime()* wird die aktuelle Systemzeit in die Variable *sysTime* eingelesen.
- Als nächstes wird mit der Funktion *sprintf()* die aktuelle Systemzeit im Format hh:mm zusammengestellt und danach als Rückgabewert geliefert. An der *Eigenschaft → Sonstige → Tooltiptext* ist eine weitere *C-Aktion* nach dem hier beschriebenen Schema erstellt. In dieser wird jedoch das aktuelle Datum zusammengestellt.
- Die vorliegende Funktion wird im Zyklus von *1s* ausgeführt.

4.10.3 Beispiel 3 – Abspielen von Sounddateien

In diesem Beispiel wird gezeigt, wie Sounddateien abgespielt werden können. Im vorliegenden Beispiel wird bei der Umschaltung zwischen den beiden Navigationsleisten *Grundlagen* und *WinCC und Windows API* eine Sounddatei abgespielt. Das hier beschriebene Beispiel ist im Bild *cc_2_keyboard_01.PDL* am *Button1* projektiert.



Projekt-Funktion CC_PlaySound()

```
#include "apdefap.h"

void CC_PlaySound(char* lpszSoundFile)
{

#pragma code("winmm.dll")
BOOL PlaySound(LPCTSTR lpszSound,HMODULE hModule, DWORD dwSound);
#define SND_FILENAME 0x00020000L
#define SND_ASYNC 0x0001
#pragma code()


BOOL 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)
{
      MessageBeep((WORD)-1);
}

}
```

- Im ersten Abschnitt wird die Datei *apdefap.h* eingebunden. Damit können aus der vorliegenden Projektfunktion heraus auch andere Projektfunktionen aufgerufen werden.
- Der Funktionskopf definiert eine Stringvariable als Übergabeparameter. Mit dieser Variable ist der Name der abzuspielenden Sounddatei zu übergeben.
- Im nächsten Abschnitt wird die Windows DLL *winmm* eingebunden. Da nur eine Funktion aus dieser DLL benötigt wird, wird diese hier direkt deklariert. Zusätzlich werden zwei symbolische Konstanten definiert.
- Die *Projekt-Funktion* geht davon aus, daß im Projektverzeichnis ein Unterverzeichnis Sound vorhanden ist. In diesem sind die im Projekt zu verwendenden Sounddateien abgelegt. Der Pfad zur gewünschten Sounddatei wird aus Projektpfad, dem Namen des Soundverzeichnisses sowie dem übergebenen Namen der Sounddatei zusammengestellt und in der Variable *szSoundPath* abgelegt.
- Mit der Windows Funktion *PlaySound()* wird die Sounddatei abgespielt. Kann die Sounddatei nicht abgespielt werden, wird stattdessen mit der Windows Funktion *MessageBeep()* ein kurzer Pipton erzeugt.

4.10.4 Beispiel 4 – Starten eines Programms

In diesem Beispiel wird die Vorgehensweise zum Starten eines Programms gezeigt. Zu diesem Zweck wird eine bereits vorhandene *Standard-Funktion* verwendet, welche das Windows API nutzt. Das Beispiel ist im Bild *cc_0_startpicture_00.PDL* projektiert.

Standard-Funktion ProgramExecute()

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

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

- Die *Standard-Funktion ProgramExecute()* leitet den übergebenen Parameter einfach an die Windows Funktion *WinExec()* weiter. Der Rückgabewert der Funktion *WinExec()* wird einfach an den Aufrufer der Funktion *ProgramExecute()* weitergeleitet. Der Start eines Programmes verlief erfolgreich, wenn der Rückgabewert größer als 31 ist.

C-Aktion am Graphik-Objekt Execute

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

      ProgramExecute("calc.exe");

}
```

- Mit der *Standard-Funktionen ProgramExecute()* wird das Programm *calc.exe* gestartet. Es handelt sich dabei um das Taschenrechnerprogramm. Es wird keine Pfadangabe gemacht, da dies für Programme, welche sich im Windows Verzeichnis befinden, nicht notwendig ist.

4.11 Standarddialoge

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema *Standarddialoge* durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_12.PDL* projektiert.

Allgemeines

Die normale Vorgehensweise bei der Erstellung eines Dialoges in WinCC besteht darin, ein WinCC Bild zu erstellen und dieses über ein *Bildfenster* anzuzeigen. Es besteht jedoch auch die Möglichkeit, in *C-Aktionen* oder anderen Funktionen Standarddialoge zu erstellen. Dabei können sowohl WinCC Standarddialoge sowie Windows Standarddialoge verwendet werden.

In diesem Kapitel wird die Verwendung einiger zur Verfügung stehender Standarddialoge gezeigt. Es ist jedoch eine Vielzahl hier nicht erwähnter Standarddialoge vorhanden. Informationen über diese finden sie im WinCC ODK sowie in der Dokumentation zum Windows API.

4.11.1 Beispiel 1 – Sprachumschaltung

In diesem Beispiel wird gezeigt, wie der WinCC Standarddialog zur Sprachumschaltung genutzt werden kann. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis → Maus → Mausklick* projektiert.

C-Aktion am Button1

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

HWND hwndParent = NULL;
DWORD dwFlags = 0;
DWORD dwSetLocaleIDs[3] = { 0x0409,0x0407,0x040C };
UINT  uSetIDArraySize = 3;
DWORD dwGetLocaleID;
BOOL bRet;
CMN_ERROR Error;

hwndParent = FindWindow(NULL,"WinCC C-Course");

//set cs language (dwGetLacaleID contains selected language ID)
bRet = DMShowLanguageDialog(hwndParent,dwFlags,dwSetLocaleIDs,
                            uSetIDArraySize,&dwGetLocaleID,&Error);

if (bRet == FALSE)
{
      printf("\r\nError in DMShowLanguageDialog()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

//set rt language
bRet = SetLanguage(dwGetLocaleID);

if (bRet == FALSE)
{
      printf("\r\nError in SetLanguage()\r\n");
      return;
}

}
```

- Im ersten Abschnitt werden die verwendeten Variablen definiert. Unter anderem wird ein Vektor bestehend aus den ID's der drei gewünschten Sprachen definiert.
- Mit der Windows Funktion *FindWindow()* wird das Fensterhandle des Runtimefensters über dessen Fenstertitel ermittelt. Es ist darauf zu achten, daß es sich bei dem in diesem Codebeispiel angegebenen Fenstertitel nicht um den standardmäßigen Titel des Runtimefensters handelt.
- Mit der API Funktion *DMShowLanguageDialog()* wird der Standarddialog zur Sprachumschaltung aufgeblendet. Der Funktion ist ein Vektor mit den ID's der im Dialog anzuzeigenden Sprachen zu übergeben. Die Funktion schreibt die ID der vom Benutzer ausgewählten Sprache in die übergebene Variable *dwGetLocaleID*.
- Der Rückgabewert der API Funktion *DMShowLanguageDialog()* wird geprüft. Dieser hat unter anderem den Wert *FALSE*, wenn der Benutzer den Dialog mit *Abbrechen* beendet.
- Mit dem hier verwendeten Dialog wird nur die CS Sprache umgeschaltet. Um auch die RT Sprache umzuschalten, ist die *Internen Funktion SetLanguage()* zu verwenden. Dieser wird die ID der im Dialog gewählten Sprache übergeben.

Dialog Sprache wählen

Bei Ausführung der zuvor erläuterten *C-Aktion* wird der nachfolgend dargestellte Dialog aufgeblendet.

4.11.2 Beispiel 2 – Variablenauswahl

In diesem Beispiel wird gezeigt, wie der WinCC Standarddialog zur Variablenauswahl genutzt werden kann. Der Inhalt der im Dialog ausgewählten Variable wird in einem *EA-Feld* angezeigt. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button2

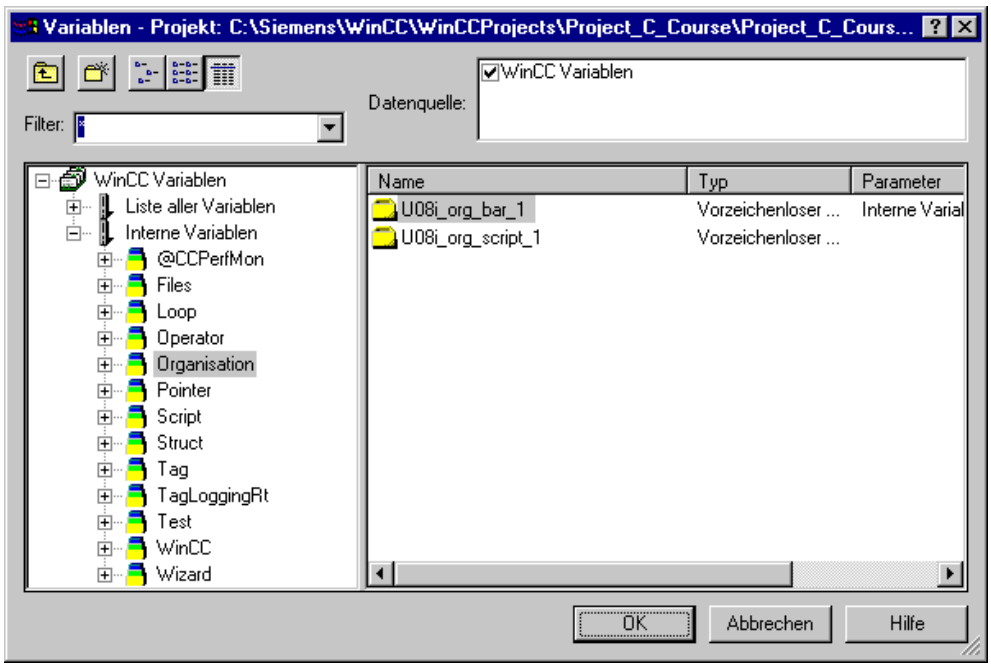
```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
//include file with the LinkType definitions
#include "trigger.h"

BOOL bRet;
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("\r\nError in DMGetRuntimeProject()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

hwndParent = FindWindow(NULL,"WinCC C-Course");

if ( DMShowVarDatabase(szProjectFile,hwndParent,NULL,NULL,
                       &dmVarKey,&Error) == FALSE)
{
      printf("\r\nError in DMShowVarDatabase()\r\n"
            "\t%s\r\n",Error.szErrorText);
      return;
}

////////////////////////////////////////////////////////////////////////////
//display tag selection
SetText(lpszPictureName,"TagName",dmVarKey.szName);

link.LinkType = BUBRT_LT_VARIABLE_DIRECT;
link.dwCycle = 0;
strcpy(link.szLinkName,dmVarKey.szName);

PDLRTSetLink(0,lpszPictureName,"TagValue","OutputValue",
             &link,NULL,NULL,&Error);

}
```

- Im ersten Abschnitt wird die Datei *trigger.h* eingebunden. In dieser ist eine in diesem Beispiel verwendete symbolische Konstante definiert.
- Im nächsten Abschnitt werden die verwendeten Variablen definiert. Unter anderem wird eine Variable *dmVarKey* zur Aufnahme von Informationen über die im Dialog selektierte WinCC Variable sowie eine Variable *link* zur Aufnahme der Informationen einer Variablenanbindung definiert.
- Mit der API Funktion *DMGetRuntimeProject()* wird der Projektname ermittelt.
- Mit der Windows Funktion *FindWindow()* wird das Fensterhandle des Runtimefensters über dessen Fenstertitel ermittelt.

- Mit der API Funktion *DMShowVarDatabase()* wird der Variablenauswahldialog geöffnet. Informationen über die im Dialog ausgewählte WinCC Variable werden in der übergebenen Variable *dmVarName* abgelegt.
- Wurde eine Variable ausgewählt, wird deren Name über einen *Statischen Text* sowie deren Inhalt über ein *EA-Feld* angezeigt.

Dialog Variablen

Bei Ausführung der zuvor erläuterten *C-Aktion* wird der nachfolgend dargestellte Dialog aufgeblendet.



4.11.3 Beispiel 3 – Fehlerbox

In diesem Beispiel wird gezeigt, wie eine Windows Fehlerbox aufgeblendet werden kann. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis* → *Maus* → *Mausklick* projektiert.

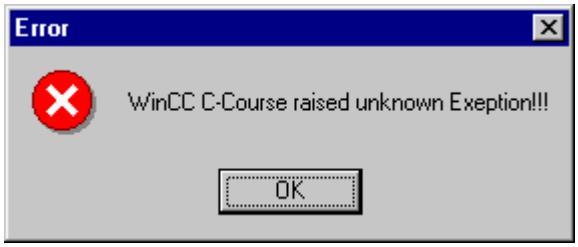
C-Aktion am Button3

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

HWND hWnd = NULL;

hWnd = FindWindow(NULL,"WinCC C-Course");

MessageBox(hWnd,"WinCC C-Course raised unknown Exeption!!!",
       "Error",MB_OK|MB_ICONSTOP|MB_APPLMODAL);

}
```

- Im ersten Abschnitt wird eine Variable *hWnd* vom Datentyp *HWND* definiert. Mit der Windows Funktion *FindWindow()* wird dieser Variable das Fensterhandle des Runtimefensters zugewiesen.
- Mit der Windows Funktion *MessageBox()* wird eine Fehlerbox geöffnet. Als zweiter Parameter ist der Fehlertext, als dritter Parameter die Überschrift der Fehlerbox anzugeben. Der vierte Parameter legt das Aussehen und das Verhalten der Fehlerbox fest. Die Fehlerbox soll nur einen *OK* Button enthalten (*MB_OK*), es soll das Fehlersymbol angezeigt werden (*MB_ICONSTOP*) und sie soll modal sein (*MB_APPLMODAL*). Damit muß der Anwender die Fehlerbox beenden, bevor er weiterarbeiten kann.
- Wurde eine Variable ausgewählt, wird deren Name über einen *Statischen Text* sowie deren Inhalt über ein *EA-Feld* angezeigt.

Fehlerbox

Bei Ausführung der zuvor erläuterten *C-Aktion* wird die nachfolgend dargestellte Fehlerbox aufgeblendet.



4.11.4 Beispiel 4 – Fragebox

In diesem Beispiel wird gezeigt, wie eine Windows Fragebox aufgeblendet werden kann und wie abhängig von dem vom Benutzer betätigten Button eine andere Aktion ausgeführt werden kann. Das Beispiel ist am nachfolgend dargestellten Objekt *Button4* am *Ereignis → Maus → Mausklick* projektiert.

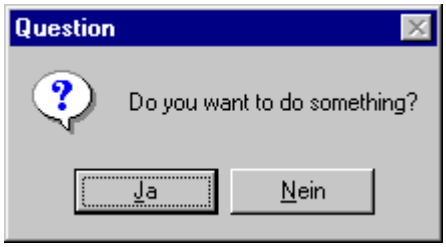
C-Aktion am Button4

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

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("\r\nExample 3\r\n");

if (iRet == IDYES)
{
      printf("User selected YES button\r\n");
}
else // if (iRet == IDNO)
{
      printf("User selected NO button\r\n");
}

}
```

- Im ersten Abschnitt wird eine Variable *hWnd* vom Datentyp *HWND* definiert. Des weiteren wird eine Variable *iRet* vom Typ *int* definiert.
- Mit der Windows Funktion *FindWindow()* wird das Fensterhandle des Runtimefensters über dessen Fenstertitel ermittelt.
- Mit der Windows Funktion *MessageBox()* wird eine Fragebox geöffnet. Der vierte Parameter der Funktion legt das Aussehen und das Verhalten der Fragebox fest. Diese soll einen *Ja* und einen *Nein* Button enthalten (*MB_YESNO*), es soll das Fragesymbol angezeigt werden (*MB_ICONQUESTION*) und sie soll modal sein (*MB_APPLMODAL*). Der Rückgabewert der Funktion wird in der Variable *iRet* abgelegt.
- Im letzten Abschnitt wird der Rückgabewert der Funktion ausgewertet. Wurde der Dialog mit *Ja* beendet, hat dieser den Wert *IDYES*, wurde dieser mit *Nein* beendet, hat dieser den Wert *IDNO*.

Fragebox

Bei Ausführung der zuvor erläuterten *C-Aktion* wird die nachfolgend dargestellte Fragebox aufgeblendet.



4.11.5 Beispiel 5 – Standarddialog Öffnen

In diesem Beispiel wird gezeigt, wie der Standarddialog zum Öffnen einer Datei aufgeblendet werden kann. Das Beispiel ist am nachfolgend dargestellten Objekt *Button5* am *Ereignis → Maus → Mausklick* projektiert.

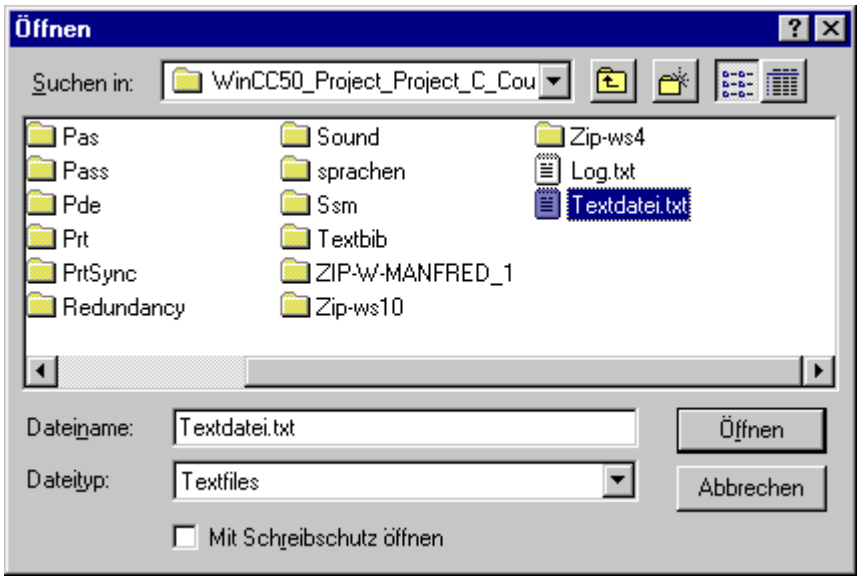
C-Aktion am Button5

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

#pragma code ("comdlg32.dll")
#include "commdlg.h"
#pragma code()

BOOL bRet;
OPENFILENAME ofn;
char szFilter[] = "Textfiles|*.txt|All Files|*.*|";
char* psz;
char szFile[_MAX_PATH+1];
char szInitialDir[_MAX_PATH+1] = "C:\\";

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = FindWindow(NULL,"WinCC C-Course");

for (psz = szFilter; *psz; psz++)
{
      if (*psz == '|')
      {
            *psz = 0;
      }
}
ofn.lpstrFilter = szFilter;

ofn.lpstrFile = szFile;
ofn.nMaxFile = _MAX_PATH+1;

GetProjectPath(szInitialDir); //if function fails initial
                              //directory is "C:\\"
ofn.lpstrInitialDir = szInitialDir;

bRet = GetOpenFileName(&ofn);

if (bRet == FALSE)
{
      printf("\r\nError in GetOpenFileName()\r\n");
      return;
}

printf("\r\nSelected File (Path+Name)\r\n%s\r\n",ofn.lpstrFile);

}
```

- Im ersten Abschnitt wird die Windows DLL *comdlg32* eingebunden.
- Im nächsten Abschnitt werden die benötigten Variablen definiert. Unter anderem wird eine Variable *ofn* vom Strukturtyp *OPENFILENAME* definiert.
- Im nächsten Abschnitt ist die Variable *ofn* mit Informationen zu füllen.
- Die Variable *ofn* wird der Windows Funktion *GetOpenFileName()* übergeben. Diese öffnet einen Dateiauswahldialog. Der Name der vom Anwender gewählten Datei wird in der Variable *ofn* gespeichert. Der Name der gewählten Datei wird ausgegeben.

Standarddialog Öffnen

Bei Ausführung der zuvor erläuterten *C-Aktion* wird der nachfolgend dargestellte Dialog aufgeblendet.



Weitere Beispiele

Die weiteren Beispiele dieses Kapitels beschäftigen sich wie Beispiel 5 mit dem Standarddateidialogen.

Im Beispiel 6 wird der Standarddialog *Speichern unter* verwendet.

Im Beispiel 7 wird die *Projekt-Funktion GetFileName()* erstellt, die den Umgang mit den Standarddateidialogen erleichtern soll. Diese Funktion kann abhängig vom einer übergebenen symbolischen Konstante einen *Öffnen* oder einen *Speichern unter* Dialog anzeigen. Zur Verfügung stehende symbolische Konstanten sind *GFN_OPEN* und *GFN_SAVE*.

4.12 Dateien

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema *Dateien* durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_13.PDL* projektiert.

Datei öffnen

Eine Datei wird in C unabhängig von ihrem Inhalt als eine Ansammlung von Zeichen betrachtet. Bevor eine Datei in einer *C-Aktion* oder einer anderen Funktion verwendet werden kann, muß diese geöffnet werden. Ist die Arbeit mit einer Datei beendet, sollte sie geschlossen werden.

Das Öffnen einer Datei erfolgt mit der Funktion *fopen()*. Im nachfolgend dargestellten Programmcode wird beispielhaft die Verwendung der Funktion *fopen()* gezeigt.

```
FILE* pFile = NULL;
pFile = fopen("C:\\Test.txt","r");
```

Um mit der Datei arbeiten zu können, muß ein Zeiger auf diese definiert werden. Zu diesem Zweck steht der Datentyp *FILE** zur Verfügung. Die Funktion *fopen()* gibt einen Zeiger auf die geöffnete Datei zurück, oder *NULL*, falls das Öffnen der Datei fehlschlägt. Als erster Parameter ist der Funktion *fopen()* der Name der zu öffnenden Datei mit Pfadangabe zu übergeben. Als zweiter Parameter ist der Modus zu übergeben, mit welchem die Datei geöffnet werden soll (z.B. zum Lesen). Die für den Modus anzugebenden Werte sind in der nachfolgenden Tabelle aufgeführt.

Modus	Beschreibung
r	Öffnen einer Datei zum Lesen. Der Rückgabewert ist *NULL*, wenn die Datei nicht existiert oder keine Leserechte bestehen.
w	Öffnen einer Datei zum Schreiben. Der Rückgabewert ist *NULL*, wenn die Datei nicht existiert oder keine Schreibrechte bestehen.
a	Öffnen einer Datei zum Anhängen an das Ende. Wenn die Datei nicht existiert, wird sie angelegt. Der Rückgabewert ist *NULL*, wenn keine Datei erstellt werden kann oder die Datei nicht beschrieben werden darf.
r+	Öffnen einer Datei zum abwechselnden Lesen und Schreiben. Der Rückgabewert ist *NULL*, wenn die Datei nicht existiert oder keine Lese- und Schreibrechte für die Datei vorhanden sind.
w+	Anlegen einer Datei zum abwechselnden Lesen und Schreiben. Existiert die Datei schon, so wird sie gelöscht. Der Rückgabewert ist *NULL*, wenn keine Rechte für diese Aktionen vorhanden sind.
a+	Öffnen einer Datei zum Lesen oder zum Anhängen an das Ende. Die Datei wird erstellt, falls sie nicht existiert. Der Rückgabewert ist *NULL*, wenn keine Lese- und Schreibrechte für die Datei vorhanden sind.

Datei schließen

Nachdem die Arbeit mit einer Datei beendet ist, sollte diese geschlossen werden. Das Schließen einer Datei erfolgt mit der Funktion *fclose()*. Im nachfolgend dargestellten Programmcode wird beispielhaft die Verwendung der Funktion *fclose()* gezeigt. Der Funktion ist der Zeiger auf die zu schließende Datei zu übergeben.

```
fclose(pFile);
```

Datei beschreiben und auslesen

Zum Beschreiben einer Datei steht eine der Funktion *printf()* ähnliche Funktion zur Verfügung. Es handelt sich dabei um die Funktion *fprintf()*. Die Verwendung der Funktion *fprintf()* erfolgt nach dem gleichen Prinzip wie die Verwendung der Funktion *printf()*. Die Ausgabe erfolgt jedoch nicht ins *Global Script Diagnosefenster*, sondern in eine Datei. Als ersten Parameter erwartet die Funktion einen Zeiger auf diese Datei. Im nachfolgend dargestellten Programmcode wird beispielhaft die Verwendung der Funktion *fprintf()* gezeigt.

```
fprintf(pFile,"%d\r\n%f\r\n",iValue,dValue);
```

Zum Auslesen einer Datei steht die Funktion *fscanf()* zur Verfügung. Die Funktion *fscanf()* ist gleich aufgebaut wie die Funktion *fprintf()*. Es sind jedoch nicht die Variablen anzugeben, deren Werte in die Datei geschrieben werden sollen. Stattdessen sind die Adressen der Variablen anzugeben, in welche die Inhalte der Datei geschrieben werden sollen.

```
fscanf(pFile,"%d\r\n%f\r\n",&iValue,&fValue);
```

4.12.1 Beispiel 1 - Daten sichern

In diesem Beispiel wird gezeigt, wie Daten in eine Datei geschrieben werden können. Die zu schreibenden Daten werden zuvor aus WinCC Variablen gelesen. Das Beispiel ist am nachfolgend dargestellten Objekt *Button1* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button1

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

FILE* pFile = NULL;

char szFile[_MAX_PATH+10];

int iData;
float fData;

//get project path
if (GetProjectPath(szFile) == FALSE)
{
      printf("\r\nError in GetProjectPath()\r\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("\r\nError in fopen()\r\n");
      return;
}

//get data to write
iData   = GetTagSDWord("S32i_course_file_1");
fData   = GetTagFloat("F32i_course_file_1");

//write data
fprintf(pFile,"%d\r\n%f\r\n",iData,fData);

fclose(pFile);

//output in diagnostics window
printf("\r\nData written in file:\r\n\t%d\r\n\t%f\r\n",
            iData,fData);

}
```

- Im ersten Abschnitt werden die benötigten Variablen definiert. Unter anderem wird eine Variable vom Typ *FILE** definiert und initialisiert.
- Mit der *Projekt-Funktion GetProjectPath()* wird der Projektpfad ermittelt.
- Als nächstes wird mit der Funktion *strcat()* der Pfad zur zu erzeugenden Datei zusammengestellt. Dieser Pfad wird der Funktion *fopen()* übergeben. Mit dieser soll die gewünschte Datei geöffnet oder angelegt werden.
- Im nächsten Abschnitt werden die zu schreibenden Daten aus den WinCC Variablen gelesen.
- Mit der Funktion *fprintf()* werden die Daten in die Datei geschrieben. Daraufhin wird die Datei wieder geschlossen.

4.12.2 Beispiel 2 - Daten lesen

In diesem Beispiel wird gezeigt, wie Daten aus einer Datei gelesen werden können. Die gelesenen Daten werden in WinCC Variablen geschrieben. Das Beispiel ist am nachfolgend dargestellten Objekt *Button2* am *Ereignis* → *Maus* → *Mausklick* projektiert.

C-Aktion am Button2

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

FILE* pFile = NULL;

char szFile[_MAX_PATH+10];

int iData;
float fData;

//get project path
if (GetProjectPath(szFile) == FALSE)
{
      printf("\r\nError in GetProjectPath()\r\n");
      return;
}

//create file name
strcat(szFile,"Data.txt");

//open file to read
pFile = fopen(szFile,"r+");

//check return value of fopen()
if (pFile == NULL)
{
      printf("\r\nError in fopen()\r\n");
      return;
}

//read data
fscanf(pFile,"%d\r\n%f\r\n",&iData,&fData);

fclose(pFile);

//set data
SetTagSDWord("S32i_course_file_1",iData);
SetTagFloat("F32i_course_file_1",fData);

//output in diagnostics window
printf("\r\nData read from file:\r\n\t%d\r\n\t%f\r\n",
            iData,fData);

}
```

- Im ersten Abschnitt werden die benötigten Variablen definiert. Unter anderem wird eine Variable vom Typ *FILE** definiert und initialisiert.
- Mit der *Projekt-Funktion GetProjectPath()* wird der Projektpfad ermittelt.
- Als nächstes wird mit der Funktion *strcat()* der Pfad zur zu öffnenden Datei zusammengestellt. Dieser Pfad wird der Funktion *fopen()* übergeben. Mit dieser soll die gewünschte Datei zum lesen geöffnet werden.
- Mit der Funktion *fscanf()* werden die Daten aus der Datei gelesen. Daraufhin wird die Datei wieder geschlossen.
- Im nächsten Abschnitt werden die gelesenen Daten in die WinCC Variablen geschrieben.

4.12.3 Beispiel 3 - Protokollieren

In diesem Beispiel wird gezeigt, wie eine Protokolldatei erstellt werden kann. Es wird eine *Projekt-Funktion* erstellt, welcher ein Protokolltext zu übergeben ist. Dieser wird in die Protokolldatei geschrieben. Das Beispiel ist am nachfolgend dargestellten Objekt *Button3* am *Ereignis → Maus → Mausklick* projektiert.

Projektfunktion LogText()

```
#include "apdefap.h"

BOOL LogText(char* lpszLogText)
{

FILE* pFile = NULL;

char szFile[_MAX_PATH+10];

//get project path
if (GetProjectPath(szFile) == FALSE)
{
      printf("\r\nError in GetProjectPath()\r\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("\r\nError in fopen()\r\n");
      return FALSE;
}

//append data
fprintf(pFile,"%s - %s\r\n",GetLocalTimeString(),lpszLogText)

fclose(pFile);

return TRUE;

}
```

- Der Funktion ist eine Stringvariable zu übergeben, welche an das Ende der Protokolldatei angehängt wird.
- Im ersten Abschnitt werden die benötigten Variablen definiert. Unter anderem wird eine Variable vom Typ *FILE** definiert und initialisiert.
- Mit der *Projekt-Funktion GetProjectPath()* wird der Projektpfad ermittelt.
- Als nächstes wird mit der Funktion *strcat()* der Pfad zur zu öffnenden Datei zusammengestellt. Dieser Pfad wird der Funktion *fopen()* übergeben. Mit dieser soll die gewünschte Datei zum anhängen geöffnet werden.
- Mit der Funktion *fprintf()* wird der übergebene Protokolltext in die Datei eingetragen. Vor jeden Protokolleintrag wird die aktuelle Systemzeit mit der *Standard-Funktion GetLocalTimeString()* eingetragen. Daraufhin wird die Datei wieder geschlossen.

C-Aktion am Button3

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

if (LogText(GetTagChar("T08i_course_file_1")) == FALSE)
{
      printf("\r\nError in LogText()\r\n");
}

}
```

- In der *C-Aktion* wird der Inhalt einer WinCC Textvariable ausgelesen und der zuvor erstellten *Projekt-Funktion LogText()* übergeben. Der Inhalt der WinCC Textvariable wird somit in die Protokolldatei eingetragen werden.

4.13 Dynamik Wizard

Im WinCC Projekt *Project_C_Course* erreichen Sie die Beispiele zum Thema *Dynamic Wizard* durch die Anwahl des nachfolgend dargestellten Icons in der Navigationsleiste. Die Beispiele sind im Bild *cc_9_example_14.PDL* projektiert.

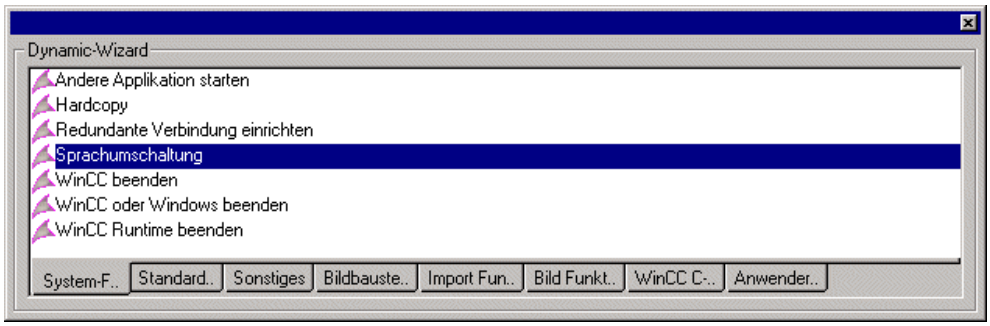
Allgemeines

Der *Dynamik Wizard* steht als zusätzliche Funktionalität im *Graphics Designer* zur Verfügung. Er kann den Anwender bei oft wiederkehrenden Projektierungsabläufen unterstützten. Damit wird der Projektierungsaufwand reduziert und mögliche Projektierungsfehler verringert.

Der *Dynamik Wizard* besteht aus verschiedenen *Dynamik Wizard* Funktionen. Es steht schon eine große Zahl an *Dynamik Wizard* Funktionen zur Verfügung. Diese können durch selbst erstellte Funktionen noch ergänzt werden.

Arbeiten mit dem Dynamik Wizard

Der *Dynamik Wizard* wird im *Graphics Designer* über den Menüpunkt *Ansicht → Symbolleisten → Dynamik Wizard* angezeigt. Nachfolgend ist der Aufbau des *Dynamik Wizards* dargestellt. Auf verschiedenen Registerkarten sind die bereits vorhandenen *Dynamik Wizard* Funktionen dargestellt. Über **D** auf die gewünschte *Dynamik Wizard* Funktion wird diese gestartet.



Eine solche *Dynamik Wizard* Funktion besteht aus verschiedenen Seiten, welche vom Anwender ausgefüllt werden müssen. Dazu gehören eine Startseite, eine Triggerseite, verschiedene Optionsseiten sowie einer Abschlußseite, welche die in den vorherigen Seiten gemachten Einstellungen zusammenfaßt.

4.13.1 Erstellung eigener Dynamik Wizard Funktionen

Zur Erstellung eigener *Dynamik Wizard* Funktionen steht ein eigener Editor zur Verfügung. Diese befindet sich im Verzeichnis *<WinCC Installationsverzeichnis>\bin*. Es handelt sich dabei um das Programm *dynwizedit.exe*.

Jede *Dynamik Wizard* Funktion ist in einer eigenen Scriptdatei abgelegt. Noch dazu ist für die Sprachen Deutsch, Englisch und Französisch jeweils eine eigene Scriptdatei vorhanden. Diese Scriptdateien sind in den nachfolgend aufgelisteten Verzeichnissen abgelegt.

- *<WinCC Installationsverzeichnis>\Wscripts\Wscripts.deu*
- *<WinCC Installationsverzeichnis>\Wscripts\Wscripts.enu*
- *<WinCC Installationsverzeichnis>\Wscripts\Wscripts.fra*

Nach dem Start des *Dynamik Wizard Editors* ist in der Symbolleiste die gewünschte Sprache auszuwählen, für welche die neue *Dynamik Wizard* Funktion erstellt werden soll.

Eine *Dynamik Wizard* Funktion muß eine vorgeschriebene Struktur haben. Im Rahmen dieses Handbuches wurden zwei Beispiele für *Dynamik Wizard* Funktionen erstellt. Die Scriptdateien, in welchen diese Beispiele vorliegen, sind im eigens zu diesem Zweck angelegten Unterverzeichnis *DynWiz* des WinCC Projekts *Project_C_Course* abgelegt. Diese Scriptdateien sind in die zuvor genannten Verzeichnisse, in denen die standardmäßig vorhandenen Scriptdateien abgelegt sind, zu kopieren. Die Beispiele können daraufhin im *Dynamik Wizard Editor* geöffnet werden.

Demo Wizard

In der Scriptdatei *Demo.wnf* ist ein *Dynamik Wizard* mit der Bezeichnung *Demo Wizard* erstellt. Dieser zeigt die grundlegenden Funktionen, welche zur Verfügung stehen, um dem Anwender komfortabel die Eingabe von Daten zu ermöglichen. Dieser *Dynamik Wizard* führt jedoch keine Aktion aus.

Motor dynamisieren

In der Scriptdatei *Motor.wnf* ist ein *Dynamik Wizard* mit der Bezeichnung *Motor dynamisieren* erstellt. Dieser wurde eigens für die Dynamisierung eines *Anwender-Objekt* namens *Motor* erstellt und kann auf keine andere Objektart angewendet werden. Das *Anwender-Objekt Motor* ist in der Projektbibliothek des WinCC Projekts *Project_C_Course* abgelegt und kann aus dieser in ein Bild eingefügt werden. Es wird über den *Dynamik Wizard Motor dynamisieren* mit einer WinCC Strurkturvariable vom Strukturtyp *Motor* verknüpft. Genauer gesagt werden verschiedene *C-Aktionen* und *Variablenanbindungen* an diesem erstellt. Es muß eine interne WinCC Textvariable *T08i_course_wiz_selected* vorhanden sein. Mit Hilfe dieser Variable kann das aktuell selektierte Motorobjekt gekennzeichnet werden.

Scriptdateien übersetzten

Eine fertig erstellte *Dynamik Wizard* Funktion ist über den Menüpunkt *Dynamik Wizard → Script übersetzen* zu übersetzen und daraufhin abzuspeichern. Um die *Dynamik Wizard* Funktion im *Graphics Designer* einsetzen zu können, muß diese in die Datenbasis des *Dynamik Wizards* eingebunden werden. Dies erfolgt über den Menüpunkt *Dynamik Wizard → Wizard Script einlesen*. Die einzulesende Scriptdatei ist aus dem erscheinenden Dialog auzuwählen.

4.13.2 Struktur einer Dynamik Wizard Funktion

Im folgenden werden die verschiedenen Abschnitte erläutert, aus denen eine *Dynamik Wizard* Funktion besteht.

Einbinden von Headerdateien und DLL‘s

Den ersten Teil einer *Dynamik Wizard* Funktion bildet die Einbindung der benötigten Headerdateien. Die wichtigste einzubindende Datei ist hier die Datei *dynamic.h*, in welcher die Funktionen unter anderem die Funktionen zur Gestaltung der Benutzeroberfläche des *Dynamik Wizards* deklariert sind. Des weiteren können hier alle gewünschten DLL’s des Windows oder des WinCC API eingebunden werden.

```
#include "dynamic.h"

#pragma code("pdlcsapi.dll")
#include "pdlcsapi.h"
#pragma code()
```

Sprachabhängige Definitionen

Soll die *Dynamik Wizard* Funktion für verschiedene Sprachen zur Verfügung stehen, ist für jede dieser Sprachen eine eigene Scriptdatei zu erstellen. Daher sollten sprachabhängige Texte vor dem Programmcode definiert werden. Somit kann eine für eine bestimmte Sprache erstellte Scriptdatei einfach kopiert werden. Es ist dann nur der Abschnitt mit den sprachabhängigen Definitionen anzupassen.

```
////////////////////////////////////////////////////////////////////////
//change only this strings to generate wizard scripts for other languages

#include  "defdeu.h"

char* DynWizGroupName     = "WinCC C-Kurs";
char* DynWizDynamicName   = "Motor dynamisieren";
char* DynWizToDoOption1   = "Wählen Sie die gewünschte Strukturvariable:";
char* DynWizGenerateInfo = "Der Motor wird mit der Strukturvariable\r\n"
                           "%s\r\ndynamisiert.\r\n";

//
////////////////////////////////////////////////////////////////////////
```

Eigenschaftsliste

Es besteht die Möglichkeit, festzulegen, für welche Objektarten eine *Dynamik Wizard* Funktion einsetzbar ist. Dies erfolgt dadurch, das eine Liste von Objekteigenschaften angegeben wird. Verfügt ein Objekt über mindestens eine der aufgeführten Eigenschaften, kann die *Dynamik Wizard* Funktion auf dieses Objekt angewendete werden. Beim *Dynamik Wizard Motor dynamisieren* wurde diese Möglichkeit genutzt, um diesen nur für *Anwender-Objekte* vom Typ *Motor* einsetzbar zu machen. Diese Objektart stellt als einzige die Eigenschaften *Hand* und *Selection* zur Verfügung. Wird eine leere Eigenschaftsliste verwendet, kann die *Dynamik Wizard Funktion* auf alle Objektarten angewendet werden. Die Eigenschaftsliste muß auf jeden Fall vorhanden sein, auch wenn diese leer ist.

```
//this wizard can only be executed on the customized object "motor"
BEGIN_PROPERTY_SCHEME
{ "Hand", VT_BOOL },
{ "Selection", VT_BOOL },
END_PROPERTY_SCHEME
```

Systemschnittstelle

Über die Systemschnittstelle werden verschiedene Eigenschaften der neuen *Dynamik Wizard* Funktion festgelegt. Die Bedeutung der einzelnen Parameter wird im Anschluß an das nachfolgende Codebeispiel erläutert.

```
BEGIN_DYNAMICS
      {
            DynWizGroupName,        //group name
            DynWizDynamicName,      //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
```

- Der erste Parameter legt fest, auf welcher Registerkate die *Dynamik Wizard* Funktion erscheinen soll
- Der zweite Parameter legt fest, unter welchem Namen die *Dynamik Wizard* Funktion erscheinen soll.
- Als dritter Parameter ist immer *NULL* zu übergeben.
- Der vierte Parameter bezeichnet den Namen des für die *Dynamik Wizard* Funktion zu verwendenden Icons.
- Als fünfter Parameter kann ein Hilfetext mit einer näheren Beschreibung der Funktionalität der *Dynamik Wizard* Funktion übergeben werden.

- Als sechster Parameter ist eine Liste mit den Namen der für die einzelnen Optionsseiten erstellten Funktionen anzugeben. Diese Liste ist mit einem *NULL* Eintrag abzuschließen. Es ist die Erstellung von maximal fünf Optionsseiten möglich.
- Als siebenter Parameter ist der Name der Bearbeitungsfunktion anzugeben, welche nach Betätigung des *Fertigstellen* Buttons aufgerufen wird.
- Als achter Parameter ist der Name der Funktion anzugeben, welche die auf den Optionsseiten gemachten Einstellungen zusammenfaßt und dem Anwender anzeigt, bevor dieser den *Fertigstellen* Button betätigt.
- Als neunter Parameter ist eine Liste der auf der Triggerseite anzuzeigenden Trigger anzugeben. Für die am häufigsten auftretenden Anwendungsfälle stehen Makros zur Verfügung, welche diese Triggerliste ausfüllen.

Globale Variablen

Für jeden in den Optionsseiten einzustellenden Parameter ist eine globale Variable zu definieren. Damit wird erreicht, daß die eingestellten Parameter in allen erstellten Funktionen bekannt sind und mit ihnen gearbeitet werden kann.

```
//global tags
char  g_MotorStructName[255] = "MotorStruct";
char* g_SelectedMotor        = "T08i_course_wiz_selected";
DWORD dwTypes[1];
```

Optionsseiten

Für jede benötigte Optionsseite ist eine eigene Funktion zu erstellen. Die Namen dieser Funktionen sind in der Systemschnittstelle anzugeben. Sie werden in der Reihenfolge, in welcher sie dort angegeben werden, aufgerufen. Zur Gestaltung der Optionsseiten steht eine große Zahl an Funktionen zur Verfügung. Die Verwendung dieser Funktionen wird im *Dynamik Wizard Demo Wizard* beispielhaft vorgeführt.

Bearbeitungsfunktion

Die Bearbeitungsfunktion ist jene Funktion, die bei Betätigung des *Fertigstellen* Buttons die eigentliche Arbeit der *Dynamik Wizard* Funktion erledigt. Der Name dieser Funktion ist in der Systemschnittstelle anzugeben. Eine umfangreiche Bearbeitungsfunktion wird im *Dynamik Wizard Motor dynamisieren* beispielhaft vorgeführt.

Infofunktion

Die Infofunktion faßt die vom Anwender gemachten Einstellungen zusammen und gibt diese Zusammenfassung auf der letzten Seite der *Dynamik Wizard* Funktion aus. Der Name dieser Funktion ist in der Systemschnittstelle anzugeben.

5 Anhang

Im Anhang finden Sie eine Sammlung der Themen die nicht direkt im *Configuation Manual* eingearbeitet wurden.

5.1 Tips und Tricks

Weitere Beispiele zur Projektierung mit WinCC.

5.1.1 Normierte Ein-/Ausgabe am EA-Feld

Damit ein EA-Feld den Wert normiert anzeigt, bzw. der eingegebene Wert normiert an die Steuerung übergeben wird, sind folgende Aktionen zu projektieren:

Aktion an Property "Ausgabewert" eines EA-Feldes (wichtig: "float", wenn Nachkommastellen erwünscht)

```
float  a;
a=GetTagFloat("DB21_DW1");
return(a/100);
```

Aktion an Ereignis "Eingabewert" eines EA-Feldes (Variable "Var1" ist ein Vorzeichenloser 16-Bit-Wert)

```
float  a;
a=GetInputValueDouble(lpszPictureName,lpszObjectName);
SetTagFloat("Var1",a*100);
```

5.1.2 Objektspezifische Aktionen bei Bildanwahl

Es gibt Anwendungsfälle, in denen Aktionen am Property eines oder mehrerer Objekte in einem Bild nur einmalig bei Bildanwahl durchzuführen sind. Eine Möglichkeit ist, eine bildspezifische Aktion am Bildobjekt unter *Ereignisse → Sonstige → Bildanwahl* zu formulieren. Dies hat aber Nachteile, daß die Aktion auf Objekte im Bild wirken muß und dadurch in der Aktion die Objektnamen fix genannt werden müssen. Die Objekte sind nicht mehr frei hantierbar. Diese Lösung ist nicht objektorientiert.

Es gibt eine Möglichkeiten, dieses Problem zu umgehen:

- Definieren Sie eine interne Variable (z.B. *dummy*), die nie aktualisiert oder gezielt gesetzt wird. Triggern Sie die Aktion am Objekt auf Änderung dieser Variable. Beim Aufschlagen des Bildes im Runtime wird die Aktion erstmalig aktiviert und würde danach nur noch bei Änderung der Variable *dummy* erneut reagieren, was aber nicht erfolgt, da diese Variable nie geändert wird.

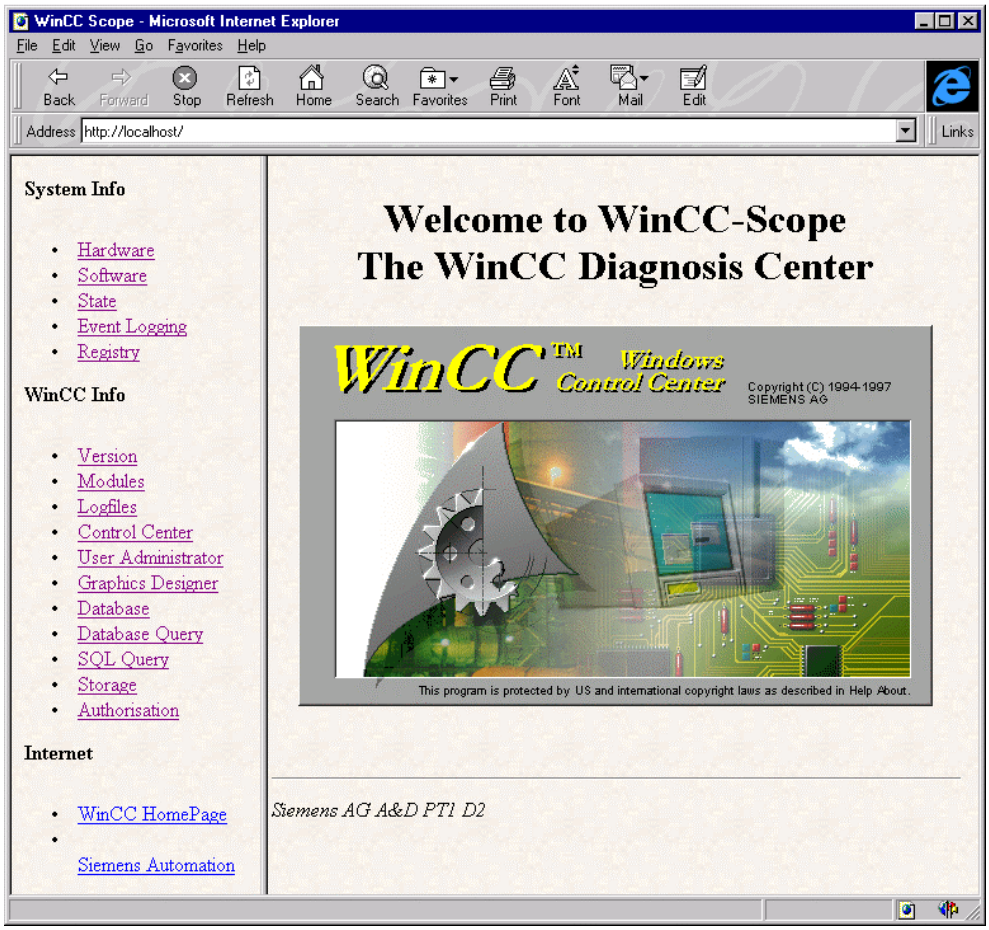
5.1.3 WinCC-Scope

Allgemeines

WinCC-Scope ist ein Werkzeug, welches Sie bei der Diagnose von WinCC-Projekten unterstützt. Es stellt Ihnen eine Vielzahl an Informationen über das aktivierte Projekt, aber auch über das jeweilige Computersystem zur Verfügung. Um mit *Scope* arbeiten zu können, wird ein Webbrowser wie der Internetexplorer benötigt. Weiters muß als Netzwerkprotokoll TCP/IP installiert sein.

Start und Bedienung

Haben Sie WinCC installiert, wurde auch standardmäßig *Scope* installiert. Bevor Scope benutzt werden kann, muß das Programm *WinCCDiagAgent.exe* gestartet werden. Es befindet sich im Verzeichniss *Siemens\WinCC\WinCCScope\bin*. Es handelt sich hierbei um einen einfachen HTTP-Server. Danach kann *Scope* über das Startmenü aktiviert werden. Auf der Startseite wird über den Verweis *How to use the new Diagnostics Interface* eine Seite mit einer allgemeinen Beschreibung über die Bedienung von *WinCC-Scope* erreicht. Klicken Sie auf den Verweiß *http://localhost*, um *Scope* zu starten. Über die Liste im linken Fenster können Informationen verschiedenster Art abgerufen werden. Im Abschnitt *System Info* werden allgemeine Informationen zum jeweiligen Computersystem erreicht, im Abschnitt *WinCC Info* werden Informationen zum gerade aktivierten WinCC-Projekt erreicht.

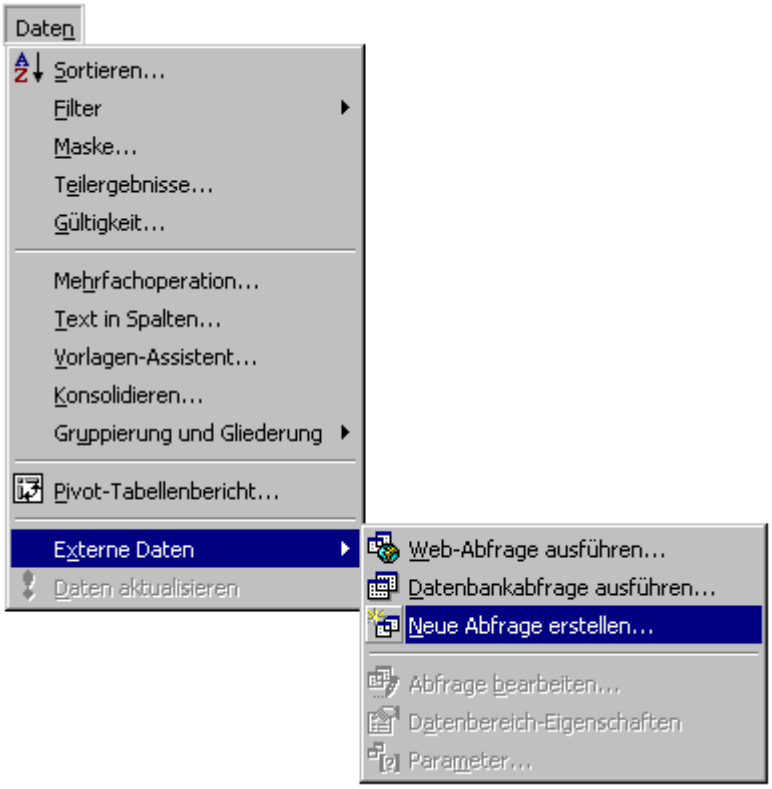


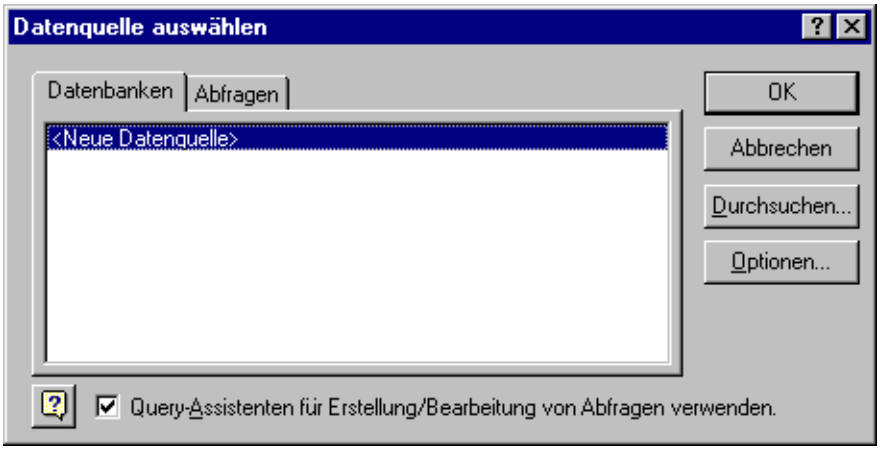
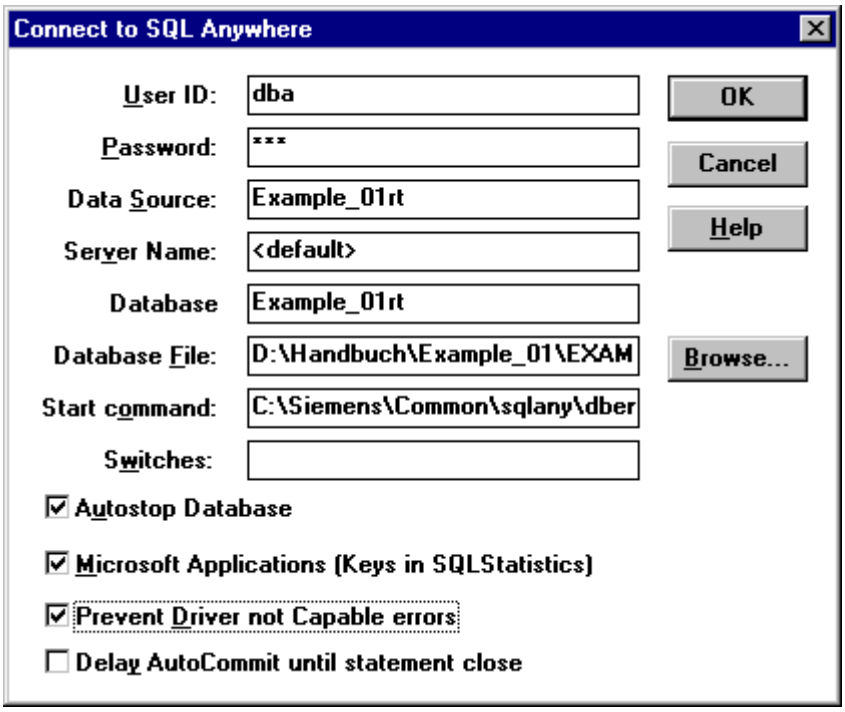
5.1.4 Datenbank-Zugriffe

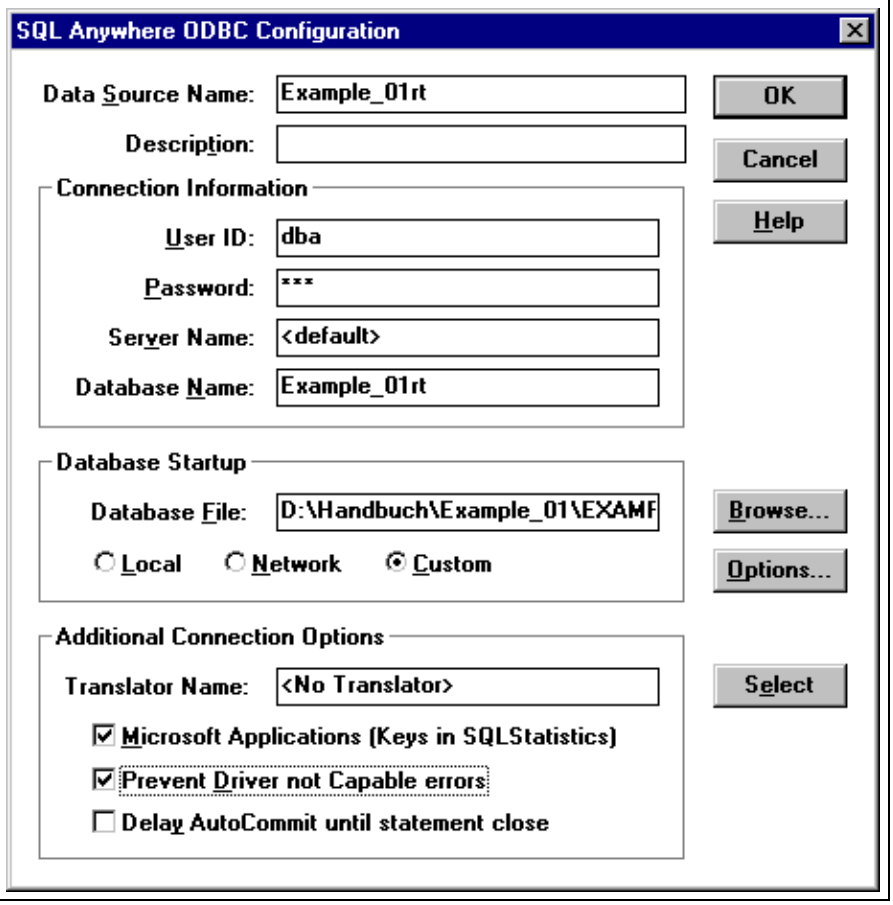
5.1.4.1 Zugriff auf die Datenbank von Excel/MSQuery aus

Die folgende Beschreibung für den Zugriff auf die WinCC-Datenbank bezieht sich auf die Verwendung von Microsoft® Excel 97 mit SR-1.

Zugriff von Excel/MSQuery

Schritt	Vorgehen: Zugriff von Excel/MSQuery
1	Excel öffnen. Über den Menüpunkt *Daten → Externe Daten → Neue Abfrage erstellen...* wird der Dialog *Datenqelle auswählen* von MSQery geöffnet. 

Schritt	Vorgehen: Zugriff von Excel/MSQuery
1	Auf der Registrierkarte *Datenbanken* den Eintrag *<Neue Datenquelle>* selektieren. Mit dem Button *OK* eine Neue Datenquelle anlegen. 
2	Im Dialog *Neue Datenquelle erstellen* den Namen der neuen Datenqelle angeben. Dieser muß nicht mit dem Namen der WinCC-Datenbank übereinstimmen. Als Treiber *Sybase SQL Anywhere 5.0* auswählen. Über den Button *Verbinden...* wird der Dialog *Connect to SQL Anywhere* geöffnet, in welchem vom Treiber benötigte Informationen einzugeben sind. Als *User ID* ist *dba* und als *Password sql* einzugeben. Über den Button *Browse* wird die zu bearbeitende Datenbank ausgewählt. Mit *OK* werden die Eingaben abgeschlossen. 

Schritt	Vorgehen: Zugriff von Excel/MSQuery
3	Ist fur die gewählte Datenbank noch keine Datenquelle konfiguriert, erscheint die Meldung *Name der Datenqelle nicht gefunden und kein Standardtreiber angegeben.* Diese Meldung bestätigen und ein weiteres mal den Button *Verbinden...* betätigen. Im Dialog *Datenquelle auswählen* die Registerkarte *Computer-Datenquelle* auswählen. Die CS- sowie die Runtimedatenbank des zur Zeit laufenden WinCC-Projekts sind in der Liste der Datenquellen bereits vorhanden. Der Name dieser Datenqellen beginnt mit der Zeichenfolge *CC_* gefolgt vom Projektname. Der Name der die Runtimedatenbank representierenden Datenqelle endet mit den Zeichen *R*. Soll jedoch eine beliebige WinCC-Datenbank bearbeitet werden, ist für diese erst eine Datenqelle anzulegen. Dies erfolgt über den Button *Neu*. Im erscheinenden Assistenten *Neue Datenquelle erstellen* auf der ersten Seite den Punkt *Benutzer-Datenquelle* auswählen und die Seite mit *Weiter* abschließen. Auf der nächsten Seite den Treiber *Sybase SQL Anywhere 5.0* auswählen und diese mit *Weiter* abschließen. Die letzte Seite mit *Fertigstellen* abschließen. Es wird der Dialog *SQL Anywhere ODBC Configuration* geöffnet, in welchem vom Treiber benötigte Informationen einzugeben sind. Als *User ID* ist wiederum *dba* und als *Password sql* einzugeben. Über den Button *Browse* wird die zu bearbeitende Datenbank ausgewählt. Mit *OK* wird der Dialog abgeschlossen. 

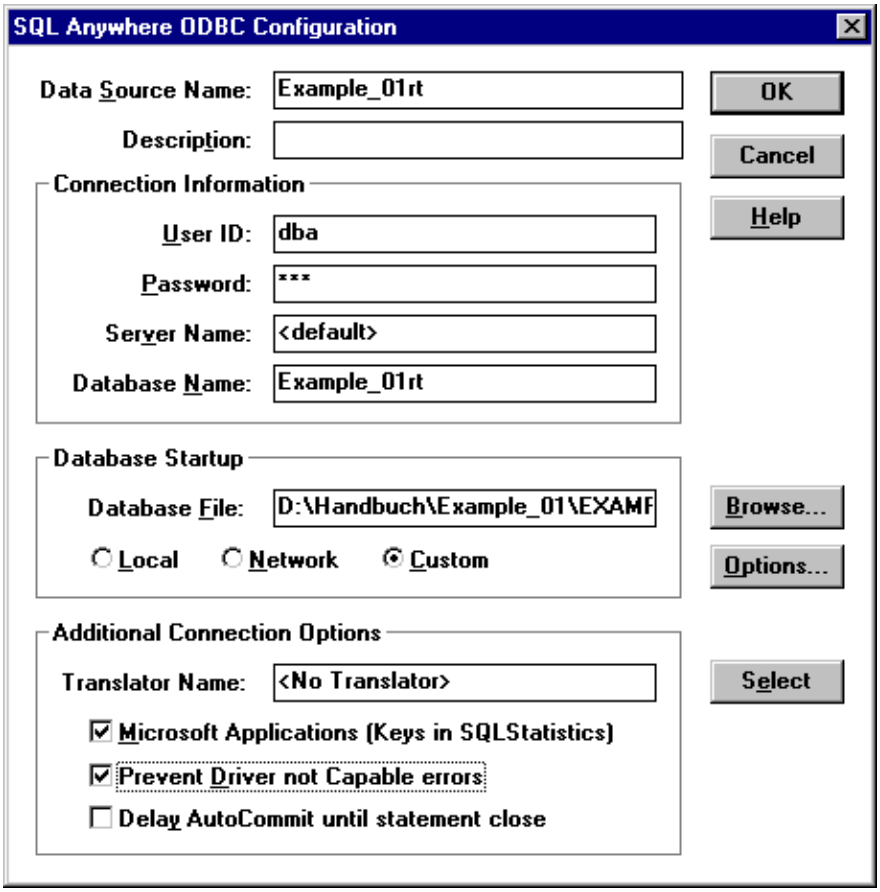
Schritt	Vorgehen: Zugriff von Excel/MSQuery
4	Die neu erstellte Datenquelle im Dialog *Datenquelle auswählen* auswählen und den Dialog mit *OK* abschließen. Den nun erscheinenden Dialog *Connect to SQL Anywhere* bestätigen. Das Konfigurieren der Datenquelle kann schon vorher über die Systemsteuerung erfolgen. Dort den *ODBC Datenquellen-Administrator* öffnen. Über den Button *Hinzufügen* wird ebenfalls der Assistent *Neue Datenquelle erstellen* geöffnet.  ODBC
5	Den Dialog *Neue Datenquelle erstellen* mit *OK* abschließen. Im Dialog *Datenquelle auswählen* die nun neu angelegte Datenquelle selektieren und den Dialog mit *OK* abschließen. Auf der ersten Seite des nun erscheinenden *Query-Assistenten* werden alle verfügbaren Tabellen und Spalten angezeigt. Die gewünschten Tabellen und Spalten auswählen und die Seite mit *Weiter* abschließen. Auf den nächsten Seiten können Filter für die Daten gesetzt sowie deren Sortierreihenfolge eingestellt werden. Auf der letzten Seite wird festgelegt, ob die Daten in Excel oder in MSQuery weiterbearbeitet werden sollen. Den Dialog mit *Fertigstellen* abschließen.
6	Im erscheinenden Dialog *Externe Daten an Excel zurückgeben* wird die Positionierung der einzufügenden Tabellen festgelegt. Weiters können die Eigenschaften des externen Datenbereichs festgelegt werden. Der Dialog wird mit *OK* abgeschlossen.

5.1.4.2 Zugriff auf die Datenbank von Access aus

Die folgende Beschreibung für den Zugriff auf die WinCC-Datenbank bezieht sich auf die Verwendung von Microsoft® Access 97 mit SR-1.

Zugriff über Access

Schritt	Vorgehen: Zugriff über Access
1	Eine Access-Datenbank öffnen oder neu erstellen. Über den Menüpunkt *Datei* → *Externe Daten* → *Importieren...* wird der Dialog *Importieren* geöffnet. Als *Dateityp* den Listeneintrag *ODBC-Datenbanken()* auswählen. Es wird automatisch der Dialog *Datenquelle auswählen* geöffnet. Auf der Registerkarte *Computer-Datenquelle* eine Datenquelle auswählen. Die CS- sowie die Runtimedatenbank des zur Zeit laufenden WinCC-Projekts sind in der Liste der Datenquellen bereits vorhanden. Der Name dieser Datenqellen beginnt mit der Zeichenfolge *CC_* gefolgt vom Projektname. Der Name der die Runtimedatenbank representierenden Datenqelle endet mit den Zeichen *R*.

Schritt	Vorgehen: Zugriff über Access
2	Ist die gewünschte WinCC-Datenbank noch nicht in der Liste aufgeführt, ist diese über den Button *Neu* erst als Datenquelle anzulegen. Im erscheinenden Assistenten *Neue Datenquelle erstellen* auf der ersten Seite den Punkt *Benutzer-Datenquelle* auswählen und die Seite mit *Weiter* abschließen. Auf der nächsten Seite den Treiber *Sybase SQL Anywhere 5.0* auswählen und diese mit *Weiter* abschließen. Die letzte Seite mit *Fertigstellen* abschließen. Es wird der Dialog *SQL Anywhere ODBC Configuration* geöffnet, in welchem vom Treiber benötigte Informationen einzugeben sind. Als *User ID* ist *dba* und als *Password sql* einzugeben. Über den Button *Browse* wird die zu bearbeitende Datenbank ausgewählt. Mit *OK* wird der Dialog abgeschlossen.  Die neu erstellte Datenquelle im Dialog *Datenquelle auswählen* selektieren und den Dialog mit *OK* abschließen.
3	Im erscheinenden Dialog *Objekte importieren* können die gewünschten Datenbanktabellen ausgewählt werden. Über *OK* werden diese in die Access-Datenbank eingefügt.

5.1.4.3 Zugriff auf die Datenbank von ISQL aus

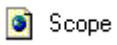
Mit Hilfe von ISQL kann direkt auf die WinCC-Datenbank zugegriffen werden. Dies geschieht aber in Eigenverantwortung, da durch bearbeiten oder löschen von Tabellen die Projektierungsdaten inkonsistent werden können.

Zugriff über ISQL

Schritt	Vorgehen: Zugriff über ISQL
1	Starten von ISQL.EXE im Verzeichnis Siemens\Common\sqlany. Es erscheint der Dialog *Interactive SQL Logon*. Als *User ID* ist *dba* und als *Password sql* einzugeben. Wird mit OK bestätigt, wird das Programm automatisch mit der aktuell geöffneten WinCC-Datenbank verbunden, und zwar mit der CS-Datenbank. Soll jedoch auf eine andere Datenbank zugegriffen werden, z.B. auf die Runtime-Datenbank, erfolgt dies über den Menüpunkt *Command → Connect*. Im erscheinenden Dialog sind wieder die gleichen eingaben für *User ID* und *Password* zu machen. Als *Database File* ist die gewünschte Datenbank mit kompletter Pfadangabe einzugeben.
2	Im Fenster Command können nun SQL-Statements eingegeben werden, welche mit dem Button *Execute* ausgeführt werden. Es folgen einige Beispiele für SQL-Statements: • select * from systable : zeigt alle Tabellennamen an • select * from <*tabellenname*> : zeigt den Inhalt der Tabelle mit dem Namen <*tabellenname*> • unload tabele <*tabellenname*> to <*filename*> : exportiert die Tabelle mit dem Namen <*tabellenname*> in die Datei mit dem Namen<*filename*> • drop table <*tabellenname*> : löscht die tabelle mit dem Namen <*tabellenname*>

5.1.4.4 Zugriff auf die Datenbank von WinCC-Scope aus

Zugriff über WinCC-Scope

Schritt	Vorgehen: Zugriff über WinCC-Scope
1	Vor dem Starten vom WinCC-Scope über das Startmenü ist die Anwendung WinCCDiagAgent.exe im Ordner *Siemens\WinCC\WinCCScope\bin* zu starten. 
2	Auf der ersten Seite über den Verweis *How to use the new Diagnostics Interface* wird eine allgemeine Beschreibung über die Bedienung von WinCC-Scope erreicht. Klicken Sie auf den Verweiß *http://localhost*, um Scope zu starten.
3	Im linken Bereich können aus einer Liste verschiedene Funktionen ausgewählt werden. • Über den Punkt *Database* werden allgemeine Informationen über die WinCC-Datenbank erreicht. • Über den Punkt *Database Query* können einzelne Tabellen einer Datenbank angezeigt werden. Als *Data Source* voreingestellt ist die CS-Datenbank des aktuell geöffneten WinCC-Projects. Der Name dieser Datenqellen beginnt mit der Zeichenfolge *CC_* gefolgt vom Projektname. Der Name der die Runtimedatenbank representierenden Datenqelle endet mit den Zeichen *R*. Es können jedoch auch andere Datenquellen angezeigt werden. • Über den Punkt *SQL Query* können auf eine zu wählende Datenquelle SQL-Statements angewendet werden. Es ist jedoch ratsam, die WinCC-Datenbank nur mit sehr guten Systemkenntnissen über SQL-Statements zu bearbeiten. Beispiele für SQL-Statements sind im vorangehenden Abschnitt *Zugriff auf die Datenbank von ISQL aus* angeführt.

5.1.4.5 Export aus Datenbank über C-Aktionen

Der Datenexport kann auch aus einem WinCC-Runtime-Bild aktiviert werden. Dazu kann eine Interaktive SQL mit Kommandozeile über ProgramExcecute gestartet werden. Die auszuführende Aktion ist in einer Kommandodatei (im Beispiel: archiv.sql) hinterlegt.

C-Aktion z.B. am Button

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

char* path =    "C:\\SIEMENS\\Common\\SQLANY\\ISQL -q -b -c";
char* parameters =
                "UID=DBA;PWD=SQL;DBN=CC_Project_97-10-21_09:53:27R";
char* action = "read D:\\WinCC\\Project\\archiv.sql";

char ExportArchive[200];

sprintf(ExportArchive,"%s %s %s",path,parameters,action);

ProgramExecute(ExportArchive);
}
```

- Die Variable *path* enthält den Pfad zum Programm ISQL.exe mit Aufrufparametern für dieses.
- Die Variable *parameters* enthält die im Dialog *Interactive SQL Logon* zu machenden Eingaben zur Verbindung zur Datenbank. Dies sind :
  - UID (User ID) : DBA
  - PWD (Password) : SQL
  - DBN (Data Base Name) : Name der ODBC-Datenquelle. Der Name dieser Datenqellen beginnt mit der Zeichenfolge *CC_* gefolgt vom Projektnamen und dem Datum und der Zeit der Projekterstellung. Der Name der die Runtimedatenbank representierenden Datenqelle endet mit den Zeichen *R*. Dieser Name kann bei aktiviertem Projekt über *Systemsteuerung → ODBC → Registerkarte Benutzer-DSN* ermittelt werden.
- Die Variable *action* gibt an, daß die in der Datei *archiv.sql* aufgelisteten SQL-Statements ausgeführt werden sollen.
- Die Anweisungen werden in *ExportArchive* zusammengefaßt und mit der Funktion *ProgramExecute()* ausgeführt.

Hinweis:
Soll aus einer anderen Datenbank als aus einer der zwei Projektdatenbanken ein Export durchgeführt werden, sollte statt dem Parameter DBN der Parameter DBF, das Datenbankfile mit Pfadangabe zur Datenbank, angegeben. Für die gerade aktivierte Projektdatenbank funktioniert dieser Weg jedoch nicht.

Inhalt der Datei archiv.sql

```
select * from PDE#HD#ProcessValueArchive#Analog;
output to D:\WinCC\Projekt\archiv.txt format ascii
```

- In der geöffneten Datenbank wird das Meßwertarchiv *pde#hd#ProcessValueArchive#Analog* selektiert und über den output-Befehl in die ASCII Datei *archiv.txt* exportiert.

5.1.4.6 Datenbank-Selektionen

Der zuvor beschriebene *select*-Befehl in der Kommandodatei selektiert Tabellen. Mit weiteren Parametern können Untermengen dieser Tabelle selektiert werden, die dann mit dem *output*-Befehl exportiert werden. Es folgen einige Beispiele zu diesem Thema.

Selektion auf einen Zeitbereich

```
select * from PDE#HD#ProcessValueArchive#Analog where T between
       '1996-5-1 10:10:0.00' and '1996-6-1 10:10:0.00'
```

Selektion ab einem Zeitpunkt

```
select * from PDE#HD#ProcessValueArchive#Analog where T >
       '1996-5-1 10:10:0.00'
```

Selektion auf Prozesswert ohne und mit Sortierung

```
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
       order by T
```

Selektion mit Auswahl der Spalten T (Time) und V (Value) auf Prozesswert

```
select T,V from PDE#HD#ProcessValueArchive#Analog where V > 100
       order by T
```

5.1.5 Serielle Kopplung

Folgende Einstellungen sind nötig, um eine serielle Kopplung aufzubauen:

Einstellungen CP525:

Nachricht:	Parameter CP525 Name:	P3964R
Prozedur:	KOMPONENTE: RK	Version : 01
Baudrate::	9600	Zeichenlänge: 8
Anzahl Stopbits: 1	Priorität: NIEDRIG	
Parität:	GERADE	

Im AG benötigt man einmal SYNCHRON im Anlaufzweig für den CP525 und SEND/RECEIVE-ALL im zyklischen Programm.

Einstellungen WinCC

Zur Optimierung sollte einer der beiden Partner die Priorität *hoch* haben, sinnvollerweise WinCC.

5.1.6 Farbtabelle

Die Farbwerte werden aus einer großen Palette zusammengesetzt.
Die 16 Grundfarben sind:

Farbe	Farbwert (Hex)	symbolische Konstante
Rot	0x000000FF	CO_RED
Dunkelrot	0x00000080	CO_DKRED
Grün	0x0000FF00	CO_GREEN
Dunkelgrün	0x00008000	CO_DKGREEN
Blau	0x00FF0000	CO_BLUE
Dunkelblau	0x00800000	CO_DKBLUE
Cyan	0x00FFFF00	CO_CYAN
Dunkelcyan	0x00808000	CO_DKCYAN
Gelb	0x0000FFFF	CO_YELLOW
Dunkelgelb	0x00008080	CO_DKYELLOW
Magenta	0x00FF00FF	CO_MAGENTA
Dunkelmagenta	0x00800080	CO_DKMAGENTA
Hellgrau	0x00C0C0C0	CO_LTGRAY
Grau	0x00808080	CO_DKGRAY
Schwarz	0x00000000	CO_BLACK
Weiß	0x00FFFFFF	CO_WHITE

Symbolische Konstante durch #define extern vordefiniert.

Mischfarben entstehen durch Zwischenwerte in der Palette.

Werden Farbänderungen mit Hilfe des Dynamik-Dialoges erstellt und die projektierten Daten mit den C-Aktionen weiterverarbeitet, können ebenfalls die Farbwerte ausgelesen werden, diese sind allerdings im dezimalen Format.

5.2 Dokumentation des S5-Meldesystems

Aufgabe und Funktion des S5-Meldesystems

Die vorliegende Dokumentation beschreibt die Funktionen und Eigenschaften der SIMATIC S5 Software:

S5-Meldesystem
Die Software dient zum zeitfolgerichtigen Erfassen von binären Meldungen, sowie deren Verarbeitung und Zwischenspeicherung. Das Programmpaket stellt innerhalb der SIMATIC S5 die notwendige Softwarefunktionalität zur Verfügung, um die Funktion 'zeitfolgerichtige Meldungserfassung' des WinCC-Systems zu realisieren.

Die prinzipielle Funktionsweise der Software läßt sich wie folgt darstellen: Die Software überwacht den binären Signalzustand von Meldungen, die der Anwender in einer Meldungsschnittstelle dem S5-Meldesystem zur Verfügung stellt. Tritt ein Wechsel in einem Signalzustand auf, so wird die Meldung mit Hilfe der Meldungsnummer identifiziert und mit dem aktuellen Datum/Uhrzeit Stempel versehen. Diese Daten werden (falls vom Anwender projektiert) um eine 32-Bit Prozeßvariable und eine alphanumerische Auftrags-/Chargenbezeichnung ergänzt. Der so gebildete Meldungsblock wird, falls notwendig, in einem Ringpuffer zwischengespeichert. Eine Zwischenspeicherung der Meldungsdaten wird immer dann benötigt, wenn mehr Meldungen pro Zeiteinheit auftreten, als über eine vorhandene BUS-Kopplung an das WinCC-System übertragen werden können. Durch diese Funktionalität wird eine zeitliche Entkopplung zwischen zeitfolgerichtiger Meldungserfassung in der SIMATIC S5 und übergeordnetem WinCC-Meldesystem erreicht und eine echtzeitfähige Meldungsverarbeitung ermöglicht.

Die vom S5-Meldesystem erzeugten Meldungsblöcke werden dem S5-Anwenderprogramm in einer Datenbaustein-Schnittstelle zur Verfügung gestellt. Durch eine vom Anwender zu realisierende S5-Kopplungssoftware werden diese Daten über eine BUS-Verbindung (z.B. SINEC H1) an das übergeordnete WinCC-Meldesystem übertragen. Dort stehen für die Meldungen umfangreiche Verarbeitungsfunktionen wie z.B. Visualisierung, Archivierung, Protokollierung, usw. zur Verfügung.

Die Projektierung des S5-Meldesystems durch den Anwender erfolgt über eine Datenbausteinschnittstelle (System-DB 80). Hier legt der Anwender den Systemrahmen fest, innerhalb der das Meldesystem arbeitet. Festlegungen der vom S5-Meldesystem benutzten Speicherbereiche, Art und Umfang der zu verarbeitenden Meldungen, sowie Einteilung der belegten Adressbereiche werden hier angegeben.

Das vorliegende Kapitel beschreibt den Einsatz und die Handhabung des S5-Meldesystems in der SIMATIC S5-Umgebung. Der Anwender erhält eine Übersicht über die von der Software verwendeten Funktions- und Datenbausteine, sowie den benötigten Speicherplatz. Es folgt eine detaillierte Schnittstellenbeschreibung aller vorhandener Datenschnittstellen zwischen dem S5-Meldesystem und dem S5-Anwenderprogramm. Als Hilfestellung ist ein Projektierungsbeispiel aufgeführt, das dem Anwender den Einstieg in die Handhabung des S5-Meldesystem erleichtern soll.

5.2.1 Auflistung der Softwarebausteine

Die SIMATIC S5 Software befindet sich auf der CD mit den Beispielen zu diesem Handbuch und ist unter dem Dateinamen *'WINCC1ST.S5D'* abgelegt.
Die Datei beinhaltet folgende Funktions- und Datenbausteine für das S5-Meldesystem:

FB	Name	Größe	Funktion
FB 80	SYSTEMFB	1114	Zeitfolgerichtiges Melden
FB 81	ANLAUFFB	135	Anlauf und Initialisierung für Zeitfolgerichtiges Melden
FB82	PCHECK	574	Aufgerufen durch FB 81
FB 83	MBLOCK	699	Aufgerufen durch FB 80
FB 84	SCHREIB	94	Aufgerufen durch FB 80
FB 87	VOLL	87	Aufgerufen durch FB 80
DB 80	System DB	512	Parametrieren des Meldesystems
Gesamt		2703	

Tabelle 1

Der Mindestspeicherbedarf hängt von der Projektierung des S5-Meldesystems ab. Folgende Datenbausteine werden immer zusätzlich benötigt.

Ringpuffer (min.)	2 DB	= 1024 Byte
Übergabefach zu WinCC	1 DB	= 512 Byte

Für jeden Offset- bzw. Parameter Datenbaustein sind weitere 512 Byte einzurechnen.

Die Berechnung der genauen Größe der Offset Datenbausteine folgt in Kapitel 5.2.3.1 Aufbau des Offset Datenbausteins, entsprechend in Kapitel 5.2.3.10 - die Größe des jeweiligen Parameter Datenbausteins.

5.2.2 Hardwarevoraussetzung

Die in Tabelle 1 für das S5 Meldesystem angegebenen Funktionsbausteine benötigen zur korrekten Ausführung folgende Hardware:

AG	CPU
AG 115U	CPU 944 * , CPU 945
AG 135U	CPU 928B
AG 155U	CPU 946/ 947, CPU 948

Tabelle 2

* nur die CPU 944 mit zwei PG-Schnittstellen besitzt eine Systemuhr

Diese CPUs verfügen über eine interne Uhr und sind somit in der Lage, die aktuelle Datum/Uhrzeit für die Bildung der Meldungsblöcke zur Verfügung zu stellen.
Über jeden eingerichteten WinCC Kanal wird ein aktuelles Datum/Uhrzeit-Telegramm zyklisch in die SIMATIC S5 CPU geschrieben. Durch den Funktionsbaustein **FB 86 : MELD:UHR** wird die interne Uhr der SIMATIC S5 mit der Uhr des WinCC-Systems synchronisiert.

5.2.3 Einbinden des S5-Meldesystems in das SIMATIC-S5 Anwenderprogramm

Um die SIMATIC S5 Software für das Meldesystem in das SIMATIC S5 Anwenderprogramm einzubinden, müssen folgende Schritte ausgeführt werden:

Alle in der Tabelle 1 angegebenen Bausteine sind von der Datei **WinCCST.S5D** auf das entsprechende AG zu übertragen.
Falls noch nicht standardmäßig implementiert, bzw. noch nicht im AG vorhanden, sind die Hantierungsbausteine für das entsprechende AG zu übertragen.

Schritt	Vorgehen: Meldesystem einbinden
1	Alle in der Tabelle 1 angegebenen Bausteine sind von der Datei **WinCCST.S5D** auf das entsprechende AG zu übertragen.
2	Falls noch nicht standardmäßig implementiert, bzw. noch nicht im AG vorhanden, sind die Hantierungsbausteine für das entsprechende AG zu übertragen.
3	Datenbaustein DB 80 gemäß Kapite5.2.5 parametrieren.
4	Datenbausteine für das Sendefach, den Ringpuffer, den Meldungs-Offset und ggf. die Meldungs-Parameter einrichten (siehe Kapitel 0).
5	Offset Datenbausteine für die verschiedenen Meldungssorten initialisieren (siehe Kapitel 5.2.3.1- Ruhezustand, Basismeldungsummer, ...).
6	Prozeßvariablen, Auftrags- und Chargenbezeichnung für die einzelnen Meldungen im Anwenderprogramm vorgeben (siehe Kapitel 5.2.3.10).
7	In den Anlauf-OBs (OB 20, OB 21, OB 22) sind folgende Bausteine aufzurufen: • **SPA HTB : SYNCHRON** (Der Hantierungsbaustein der jeweiligen CPU) • **SPA FB 81 : ANLAUFFB**
8	Im OB 1 sind folgende Bausteine aufzurufen: • für die zyklische Bearbeitung der Meldungen **SPA FB 80 : SYSTEMFB** • ein vom Anwender erstellter Funktionsbaustein zum Übertragen der Meldungsblöcke an das überlagerte WinCC-System (siehe Kapitel 5.2.4.4)
9	Weitere Funktionalitäten sind gemäß folgender Kapitel einzufügen: • Die Synchronisation von Datum und Uhrzeit durch FB 86 : MELD:UHR (siehe Kapitel 5.2.7).

Tabelle 3

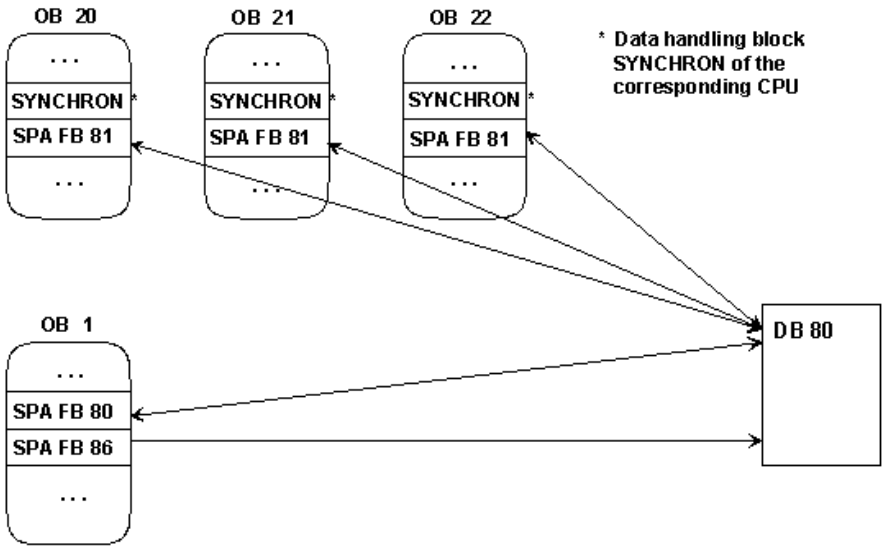


Abbildung 1

Allgemeine Beschreibung des S5-Meldesystems

Folgende Komponenten des S5 - Medesystems werden beschrieben:

- Offset-Datenbaustein
- Parameter-Datenbaustein
- Meldungsblock
- Ringpuffer
- Sendefach
- Systemdatenbaustein

Zusammenhang zwischen den einzelnen Komponenten:

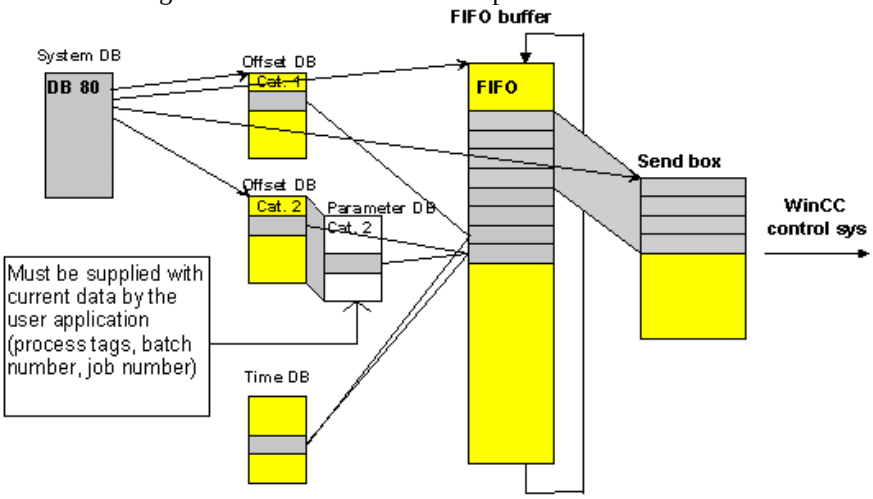


Abbildung 2

Bevor das Meldungserfassungssystem Meldungen überwachen und erfassen kann, müssen diese in den entsprechenden Datenbausteinen projektiert werden. Es werden vier Meldungssorten unterschieden:

Sorte	Definition: Meldesorten
1	Meldung ohne Parameter
2	Meldung mit Prozeßvariablen (2 DW)
3	Meldung mit Prozeßvariablen (2 DW) und Auftrags-/Chargenbezeichnung (3 DW)
4	Meldung mit Prozeßvariablen (2 DW) und Auftrags-/Chargenbezeichnung (3 DW) und Reserve (3 DW)

Tabelle 4

Für das Meldesystem, kann für die Bildung von Meldeblöcken ein Datum-/Uhrzeit-Stempel global mit angegeben werden. Bei fehlendem Datum/Uhrzeit Stempel ergänzt das WinCC-System die Meldungsblöcke (siehe Kapitel 5.2.3.11) mit den entsprechenden Informationen.

5.2.3.1 Aufbau des Offset Datenbausteins

Der Offset Datenbaustein ist für alle vier Meldungssorten gleich aufgebaut. Die entsprechende Datenbaustein-Adresse wird für jede benötigte Meldungssorte (siehe 5.2.3.1) im System Datenbaustein DB 80 angegeben.

Offset Datenbaustein für die entsprechende Meldungssorte:

DW	Inhalt	Zuordnung
DW 0	Frei	Kopf
DW 1	Basismeldungsnummer	
DW 2	Adresse des letzten Signalzustandsblocks	
DW 3	Frei	
DW 4	Signalzustände der Meldungen - Bit Nr.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Signalzustandsblock 1
DW 5	Ruhezustandsbits	
DW 6	Quittierungsbits	
DW 7	Flankenmerker	
DW 8	Signalzustände der Meldungen - Bit Nr.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Signalzustandsblock 2
DW 9	Ruhezustandsbits	
DW 10	Quittierungsbits	
DW 11	Flankenmerker	
DW 12	Signalzustände der Meldungen - Bit Nr.: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Signalzustandsblock 3
DW 13	Ruhezustandsbits	

Tabelle 5

Nachfolgend wird beschrieben:

- Basismeldungsnummer
- Offsetmeldungsnummer
- Signalzustandsblock
- Adresse des letzten Signalzustandsblocks
- Signalzustände
- Ruhezustandsbits
- Quittierungsbits
- Flankenmerker

5.2.3.2 Basismeldungsnummer

Jeder Meldung ist eine bestimmte Meldungsnummer zugewiesen, mit der man die auftretenden Meldungen unterscheiden kann. Die Meldungsnummer setzt sich aus der Basismeldungsnummer und einer Offsetmeldungsnummer zusammen.
Für jede benutzte Meldungssorte muß eine unterschiedliche Basismeldungsnummer angegeben werden. Fortlaufend von dieser Basismeldungsnummer, werden die Meldungen dieser Sorte durch die Offsetmeldungsnummer unterschieden.
Die Basismeldungsnummer für die entsprechende Meldungssorte wird in DW 1 des zugehörigen Offset Datenbausteins angegeben (siehe 5.2.3.10).

Sonderfall

Bei Benutzung der Meldungssorte 1 ist es möglich zwei Offset Datenbausteine zu verwenden. Um eine fortlaufende Meldungsnummerierung dieser Meldungssorte zu erreichen, ist die Basismeldungsnummer des zweiten Offset Datenbausteins auf die Basismeldungsnummer des ersten Offset Datenbausteins plus dessen Meldungskapazität (1008 Meldungen) einzugeben.

Berechnung der Meldungsnummer:

Meldungsnummer = Basismeldungsnummer + Offsetmeldungsnummer

Beispiel:

Berechnung	Beschreibung:
Gegeben:	Meldungssorte 1, fortlaufende Meldungsnummerierung Beginn : Meldungsnummer 10000
Gesucht:	Basismeldungsnummer der beiden Offset Datenbausteine
10000	Basismeldungsnummer des ersten Offset Datenbausteins:
10000 + 1008 = 11008	Basismeldungsnummer des zweiten Offset Datenbausteins:

5.2.3.3 Offsetmeldungsnummer / Signalzustände der Meldungen

Die Signalzustände der Meldungen befinden sich in den Offset Datenbausteinen der entsprechenden Meldungssorte an der jeweiligen Bitposition der Offsetmeldungsnummer.
Die Offsetmeldungsnummer der entsprechenden Meldung ergibt sich beginnend mit den 16 Bits (Bit 0-15) von DW 4. Die fortlaufende Nummerierung erfolgt in '4-er' Schritten (DW8, DW12, ...).

Signal-zustandsblock	**Signalzustandsblock beginnt bei Datenwort**	**Bitnummer 0 - 15 entspricht Offsetmeldungsnummer**
1	4	0 - 15
2	8	16 - 31
3	12	32 - 47
4	16	48 - 63
...		...
62	248	976 - 991
63	252	992 - 1007

Tabelle 6

Berechnung der Offsetmeldungsnummer:

Offsetmeldungsnummer = Meldungsnummer - Basismeldungsnummer
Offsetmeldungsnummer = (Datenwort / 4 -1) * 16 + Bit Nr. (0-15)
Offsetmeldungsnummer = (Signalzustandsblock - 1) * 16 + Bit Nr. (0-15)

Berechnung von DB, DW, Bit Nr. aus der Offsetmeldungsnummer:

Datenbaustein = Offset Datenbaustein
Datenwort = (Offsetmeldungsnummer / 16+ 1) * 4
Bit Nr. = Offsetmeldungsnummer % 16

Bei Meldungssorte 1 kann sich ein Datenwort größer 252 ergeben, dann gilt:

Datenbaustein = Offset Datenbaustein + 1
Datenwort = Datenwort - 252
Bit Nr. = Bit Nr.

Beispiel 1:

Gegeben: DW 248, Bit 7, Basismeldungsnummer = 10000
Gesucht: Meldungsnummer

Signalzustandsblock = 248 / 4
= 62
Offsetmeldungsnummer = (Signalzustandsblock - 1) * 16 + Bitnummer
= (62 - 1) * 16 + 7 = 983
Meldungsnummer = Basismeldungsnummer + Offsetmeldungsnummer
= 10000 + 983 = 10983

Die gewünschte Meldungsnummer ist 10983.

Beispiel 2:

Gegeben: Meldungssorte 1 mit zwei Offset Datenbausteinen,
Meldungsnummer = 12000, Basismeldungsnummer = 10000
Gesucht: DB, DW, Bit Nr.

Offsetmeldungsnummer = Meldungsnummer - Basismeldungsnummer
= 12000 - 10000 = 2000
Bit Nr. = Offsetmeldungsnummer % 16 = 0
Datenwort = (Offsetmeldungsnummer / 16 + 1) * 4
= (2000 / 16 + 1) * 4 = 504

Das Datenwort ist größer 252.

Datenbaustein = Offset Datenbaustein + 1
Datenwort = 504 - 252 = 252
Bit Nr. = 0

Die Meldungsnummer 12000 ist im zweiten Offset Datenbaustein der 1. Meldungssorte, in Datenwort 252 Bit Nr. 0 zu finden.

5.2.3.4 Signalzustandsblock

Der erste Signalzustandsblock beginnt bei Datenwortadresse 4, Die nachfolgenden Signalzustandsblöcke folgen in einem Abstand von 4 Datenworten (DW 8, DW 12, ...). Siehe hierzu auch Tabelle 5 bzw. Tabelle 6.

Für jeden Offset Datenbaustein sind 63 Signalzustandsblöcke möglich (Signalzustandsblock 1 bis 63).

Ein Signalzustandsblock beinhaltet 16 Signalzustände. Somit ergeben sich 63 * 16 = 1008 mögliche Meldungen in einem Offset Datenbaustein.

Aufbau des Signalzustandsblocks:

DW	Bitnummer	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Signalzustände	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Ruhezustände	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	Quitierungsbits	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Flankenmerker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 7

Weitere Informationen über diese vier Bitzustände folgen in diesem Kapitel.

Berechnung des jeweiligen Signalzustandsblocks:

Signalzustandsblock = (Offsetmeldungsnummer / 16) + 1
Signalzustandsblock = Datenwort / 4

Berechnung des Datenwortes mit dem der jeweilige Signalzustandsblock beginnt:

Erstes Datenwort des Signalzustandsblocks = Signalzustandsblock * 4

5.2.3.5 Adresse des letzten Signalzustandsblocks

Durch die Angabe der DW-Adresse, des letzten mit Meldungen belegten Signalzustandsblocks, wird die Anzahl der möglichen Meldungen der entsprechenden Meldungssorte angegeben.

Berechnung des letzten Signalzustandsblocks:

letzter Signalzustandsblock = benötigte Meldungen dieser Meldungssorte / 16

```
// Incompletely filled (16 messages) signal condition block
if ((required messages of this message category % 16) != 0)
{
        ++ last signal message block;
}
```

Bei Meldungssorte 1 kann ein Meldungsvolumen größer 1008 Meldungen auftreten, dann gilt:

1. Offset DB:

letzter Signalzustandsblock = 63
Adresse des letzten Signalzustandsblocks = 63 * 4 = 252

2. Offset DB:

letzter Signalzustandsblock = (benötigte Meldungen dieser Meldungssorte - 1008) / 16

```
// Incompletely filled (16 messages) signal condition block
if (((required messages in this message category – 1008) % 16) != 0)
{
        ++ last signal message block;
}
```

Berechnung der DW-Adresse des letzten Signalzustandsblocks:

DW-Adresse des
letzten Signalzustandsblocks = letzter Signalzustandsblock * 4

Beispiel:

Gegeben: 1030 Meldungen der Meldungssorte 1

1. Offset DB:

Adresse des letzten Signalzustandsblocks = 63 * 4 = 252

2. Offset DB:

benötigte Meldungen - 1008 = 1030 - 1008 = 22
(benötigte Meldungen - 1008) / 16 = 22 / 16 = 1
(benötigte Meldungen - 1008) % 16 = 22 % 16 = 6
Letzter Signalzustandsblock = 2
Adresse des letzten Signalzustandsblocks = 2 * 4 = 8

5.2.3.6 Signalzustände

Position: 1. Datenwort des Signalzustandsblocks (siehe Tabelle 5).

Der Anwender hat dafür zu sorgen, daß die Signalzustände der entsprechenden Meldungen in den dafür vorgesehenen Datenwörtern der Offset Datenbausteine der entsprechenden Meldungssorte eingetragen werden. Dies kann durch kontinuierlich prozeßbegleitende Signal-Aktualisierung durch das Steuerungsprogramm geschehen.

5.2.3.7 Ruhezustände

Position: 2. Datenwort des Signalzustandsblocks (siehe Tabelle 5).

Unter dem Ruhezustand eines Signals versteht man den Signalpegel im passiven Betriebszustand. Damit wird festgelegt, ob ein Signal (Meldung) 'low'- oder 'high'-aktiv arbeitet. Diese Information wird benötigt, um herauszufinden, ob eine Meldung 'kommend' oder 'gehend' ist.
Besitzt eine Ereignisänderung den negierten Zustand bezüglich des Ruhezustandes, handelt es sich um eine 'kommende' Meldung. Bei einer 'gehenden' Meldung gleicht der Zustand der Ereignisänderung dem des zugehörigen Ruhezustandes.

Die Ruhezustände der Meldungen müssen vom Anwender an den entsprechenden Positionen angegeben werden.

5.2.3.8 Quittierungsbits

Position: 3. Datenwort des Signalzustandsblocks (siehe Tabelle 5).

Quitterungsbits werden nicht projektiert, sondern innerhalb des laufenden Programms ausgewertet. Hierbei werden die Meldungen direkt von dem überlagerten PC entsprechend der jeweiligen Quittierungsphilosophie quittiert. Diese meldungsbezogenen Quittierungen sendet der PC an das betreffende AG mit den dazu projektierten Meldungen des integrierten Meldungserfassungssystems.

Das entsprechende Quittierungsbit wird vom S5 Meldesystem einen AG Zyklus lang gesetzt.
Das Anwendungsprogramm muß diese Information entsprechend auswerten.

5.2.3.9 Flankenmerker

Position: 4. Datenwort des Signalzustandsblocks (siehe Tabelle 5).

Die Flankenmerker dienen zur Feststellung aufgetretener Ereignisänderungen (Meldungsänderung). Sie werden nicht projektiert sondern innerhalb des S5-Meldesystems ausgewertet.

5.2.3.10 Aufbau des Parameter Datenbausteins

Für die Meldungssorte 2 bis 4 müssen neben einem sogenannten Offset Datenbaustein noch Parameter Datenbausteine für zusätzliche Daten der jeweiligen Meldung projektiert werden. Der Signalzustand einer Meldung ist im Offset Datenbaustein untergebracht. Die Adressen der Parameter Datenbausteine sind in fortlaufenden Datenbausteinen abgelegt und schließen sich direkt an den zugehörigen Offset Datenbaustein an.

Zusammenhang zwischen Offset- und Parameter Datenbaustein:

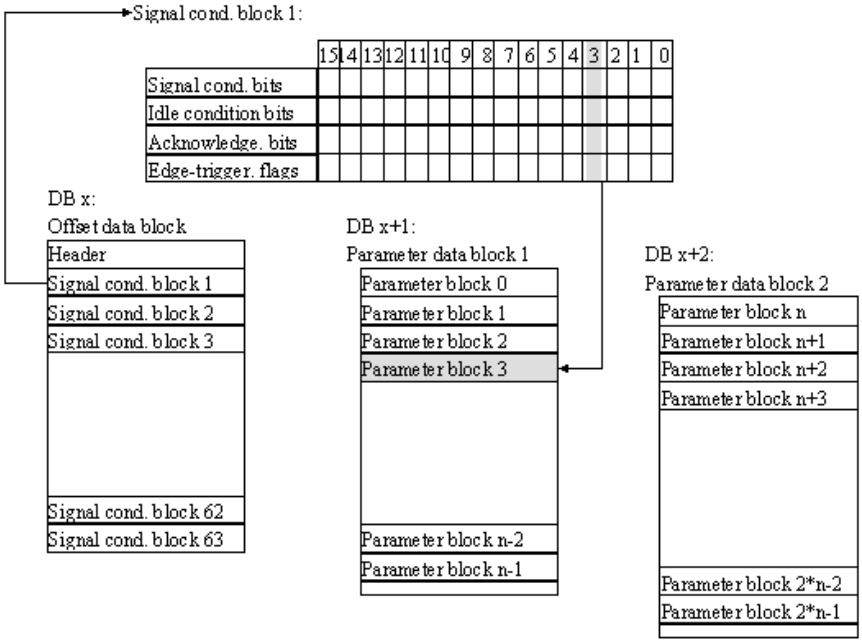


Abbildung 3

Sorte	max. Anzahl	Größe des Parameter-Blocks	Anzahl Blöcke je Parameter DB	max. Anzahl Parameter DBs
1	1008 / 2016	-	-	-
2	1008	2 DW	128	8
3	1008	5 DW	51	20
4	1008	7 DW	36	28

Tabelle 8

Berechnung der Anzahl Parameter Datenbausteine:

$$\text{Number of Parameters Data Block} = \frac{\text{used Messages}}{\text{Number of parameter blocks per parameter data block}}$$

Bei der Projektierung ist darauf zu achten, daß es keine Adressüberschneidungen mit Datenbausteinen einer anderen Meldungssorte auftreten und die Anzahl der Parameter Datenbausteine auch für den möglichen zukünftigen Ausbau vorgesehen ist.
Ein Parameterbaustein enthält Parameterblöcke, die den einzelnen Meldungen zugewiesen sind. Die Parameterblöcke sind, beginnend mit dem Parameterblock für die erste Meldung dieser Meldungssorte, fortlaufend im Parameterbaustein abgelegt. Die Parameterblöcke werden über die Grenzen des Parameter-DB hinweg fortlaufend hochgezählt. Bei erreichen des Parameter-DB Ende wird der Parameterblock mit der nächsten Nummer ab DW 0 des folgenden Parameter-DB weitergeführt. Es werden immer nur ganze Parameterblöcke im Parameter Datenbaustein abgelegt.

Berechnung der Anfangsadresse eines Parameterblocks:

Offsetmeldungsnummer	=	Meldungsnummer - Basismeldungsnummer
Parameter DB	=	Offset DB + 1 + (Offsetmeldungsnummer / Parameterblöcke je Parameter DB)
Anfangsadresse Parameter DB	=	(Offsetmeldungsnummer % Parameterblöcke je Parameter DB) * Größe Parameterblock

Der Anwender hat dafür zu sorgen, daß die entsprechenden Daten (Prozeßvariablen, Auftragsnummer, Chargenbezeichnung) an der entsprechenden Adresse zur Verfügung stehen.

5.2.3.11 Aufbau eines Meldungsblocks

Ein Meldungsblock, der an das überlagerte **WinCC**-System gesendet wird, besteht aus mehreren aneinander gereihten Datenwörtern. Diese enthalten alle meldungsspezifische Informationen. Die Summe der Datenwörter ergeben einen Meldungsblock. Die Größe der Meldungsblöcke differiert zwischen den einzelnen Meldungssorten (siehe Tabelle 10).

Ein Meldungsblock besteht unabhängig von der Meldungssorte immer aus mindestens zwei Datenwörtern. Dies sind die Meldungsnummer und der Meldungsstatus. Abhängig davon, ob die Meldungen mit Datum und Uhrzeit (3 Datenworte) und entsprechenden Parametern versehen werden, kann ein Meldungsblock die maximale Größe von 12 Datenwörtern besitzen.

DW	Beschreibung:
1.DW	Meldungsnummer
2. DW	Meldungsstatus
3. DW	Uhrzeit
4. DW	Uhrzeit
5. DW	Datum
6. DW	Prozeßvariable
7. DW	Prozeßvariable
8. DW	Auftragsnummer
9. DW	Auftragsnummer
10.DW	Chargenbezeichnung
11. DW	Reserve
12. DW	Reserve

Tabelle 9

Werden die Meldungen ohne Datum/Uhrzeit Stempel versehen, entfallen die dafür nötigen drei Datenwörter an der vorgesehenen dritten bis fünften Position des Blockes. Die Parameterdatenworte werden dann lückenlos an das Status-Datenwort angehängt. Die jeweilige Größe eines Meldungsblockes (DW-Anzahl) ist je nach Meldungssorte und gewünschtem Datum/ Uhrzeit Stempel verschieden und kann Tabelle 10 entnommen werden.

Bestimmung der Meldungsblocklänge in Abhängigkeit der Meldungssorte:

Sorte	Meldungsblocklänge in DW ohne Datum und Uhrzeit	Meldungsblocklänge in DW mit Datum und Uhrzeit
1	2	5
2	4	7
3	7	10
4	9	12

Tabelle 10

5.2.3.12 Meldungsnummer

Jeder Meldung ist eine bestimmte Meldungsnummer zugewiesen, mit der sie eindeutig identifiziert werden kann.

5.2.3.13 Meldungsstatus

Der Meldungsstatus ist wie folgt aufgebaut:

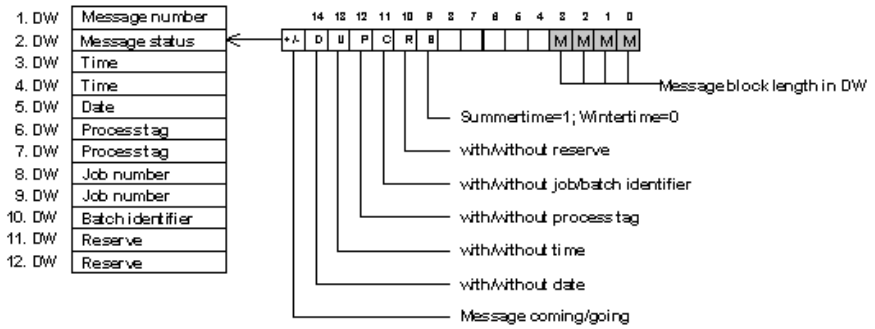


Tabelle 11

5.2.3.14 Datum-/ Uhrzeit Stempel

Das Datum und die Uhrzeit wird vom Funktionsbaustein **FB 86 : MELD:UHR** im Dual-Code zur Verfügung gestellt: Siehe Kapitel 5.2.9 Aufgabe und Funktion der S5-Uhrzeitsynchronisation.

5.2.3.15 Prozeßvariable

Zwei Datenworte, über die Prozeßvariablen bei eintreffen einer Meldung festgehalten und an das ProzessSystem übergeben werden können.

5.2.3.16 Auftragsnummer / Chargenbezeichnung

Die ersten beiden Datenworte, sind abhängig von der Projektierung, als vorzeichenbehaftete 32-Bit Dualzahl oder als insgesamt vier ASCII-Zeichen zu interpretieren. Das dritte Datenwort ist als zwei ASCII-Zeichen zu interpretieren.
Über diese drei Datenworte kann dem **WinCC**-System die aktuelle Auftragsnummer bzw. Chargenbezeichnung bei eintreffender Meldung übergeben werden.

5.2.3.17 Reserve

Die zwei Reserve-Datenworte der Meldungssorte 4 sind für zukünftige Erweiterungen vorgesehen, aber momentan noch nicht im **WinCC**-System implementiert.

5.2.3.18 Bildung eines Meldeblocks

Nachdem eine Meldung erkannt wurde, wird durch die aktuell überprüfte Bitposition die entsprechende Meldungsnummer ermittelt und als erstes Datenwort des Meldungsblockes in den Ringpuffer abgelegt. Je nach kommender oder gehender Meldung, Sorte und Wunsch auf Datum- und Uhrzeitstempel. Wird die entsprechende Statusmaske gewählt und als zweites Datenwort des Meldungsblockes in den Ringpuffer abgelegt. Ist das entsprechende Bit in dem System-Datenbaustein für einen Datum- und Uhrzeitstempel parametriert worden, folgen nun die drei Datenwörter, welche im System Datenbaustein 80 ab Adresse DW 190 im geforderten PC-Format zur Verfügung stehen. Je nach Meldungssorte wird ggf. der zugehörige Parameterblock aus dem entsprechenden Datenarchiv (Parameter Datenbaustein) gelesen und zum Fertigstellen des Meldungsblockes in dem Ringpuffer der letzten Eingabe ergänzt. Daraufhin wird das nächste Status-Bit der folgenden Meldung untersucht. Dies wird solange durchgeführt, bis alle parametrierten Meldungen abgearbeitet wurden.

5.2.3.19 Der interne Ringpuffer (FIFO)

Ein Ringpuffer ist ein Speicher, an dessen Ende wieder sein Anfang folgt, d.h. der Speicherbereich wird bildhaft zu einem Ring geschlossen. Damit wird erreicht, daß der Speicher einerseits von seiner Größe begrenzt wird und andererseits durch seinen jeweiligen Neubeginn an dessen Anfang nicht endlich ist. Im Meldungserfassungssystem hat dies zur Folge, daß bei dem Erreichen des virtuellen Endes ohne Entnahme der vorhergehenden Daten (Puffer = voll), die ältesten Daten durch die neusten überschrieben werden und somit als Information verloren gehen. Der Ringpuffer im RAM dient, wie es der Name bereits sagt, als Puffer für die erfaßten Meldungen, bevor diese an den PC weitergeleitet werden. Im RAM besteht der Ringpuffer aus einem Speicherbereich von mindestens zwei Datenbausteinen und kann, je nach Parametrierung, im Rahmen der maximal zulässigen Datenbausteine eines Automatisierungsgerätes bzw. noch frei verfügbaren DBs des Anwenderprogramms, beliebig groß gewählt werden. Der Anwender teilt dem Meldesystem eine ihm für die Archivierung zur Verfügung stehende Anzahl von Datenbausteinen mit. Bei mehreren Datenbausteinen ist es Voraussetzung, Datenbausteine mit fortlaufenden DB-Nummern zu verwenden. Somit wird vom Anwender im System-DB als Parameter die Anfangs-DB-Nummer und die Nummer des End-DB's des Puffers angegeben. Alle Datenbausteine, die sich wertmäßig zwischen Anfangs-DB und End-DB befinden (inklusive der beiden Datenbausteine), gehören dem Puffer als Speicherplatz an.

5.2.3.20 Das Sendefach - Datenübertragung zum überlagerten WinCC-System

In den internen Ringpuffer werden vom S5 Meldesystem zunächst grundsätzlich alle Meldungseinträge eines jeden laufenden Zyklus eingeschrieben. Die Meldungseinträge (bis zu max. einem Datenbaustein Inhalt) werden nach abgeschlossener Erfassung in die Meldeschnittstelle (Sendefach), sofern diese bereit ist, Daten aufzunehmen, transferiert. Die Meldeschnittstelle in Form eines Datenbausteins dient für die Übertragungs-funktionsbausteine (STEP 5 - Hantierungsbausteine) als Datenquelle. Die Hantierungsbausteine bilden eine Schnittstelle zu dem entsprechenden Kommunikationsprozessor für den eingesetzten Prozeßbus (z.B. für SINEC-H1 Bus).

Aufbau des Sendefaches:

DW	Inhalt
DW 0	Länge des Datenblocks
DW 1	KY = [AG-Nr.] , [CPU-Nr.]
DW 2	KY = [0] , [Anzahl der Meldungen]
DW 3	Beginn der Nutzdaten (Meldungsblöcke)

Tabelle 12

DW 0:

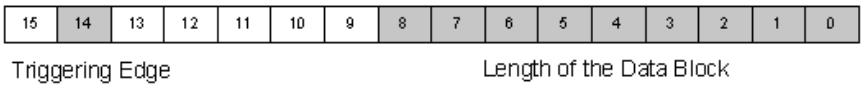


Tabelle 13

DW 0 des Sendefaches ist erstens durch das Bit Nr. 14 die Anstoß-Flanke eines gewünschten Auftrages zu entnehmen und zweitens aus den Bits Nr. 0-8 die Quelldaten-Länge zu ermitteln.

Da der zu übertragene Datenblock maximal 256 Datenwörter lang sein kann, mit einem Byte aber nur eine Zahl bis 255 darstellbar ist, ist die getrennte Abfrage der Bytes über die Befehle DL bzw. DR nicht möglich. Deshalb wird empfohlen, das DW 0 in einen Hilfsmerker zu transferieren. Dies bietet zugleich den Vorteil, daß das Freigabebit einzeln direkt auswertbar ist. Diese Operation kann bei der Verwendung von Datenwörtern nicht angewandt werden.

Bei erfüllter Bedingung sollte das Bit, welches als Flanke für einen einmaligen Anstoß eines Sendeauftrags dient, wieder zurückgesetzt werden. Die dann verbleibenden gesetzten Bits entsprechen sogleich der übertragenen Quelldaten-Länge und können in den Datenbereich der indirekten Parametrierung als QLAE geschrieben werden.
Nach einem erfolgreich abgeschlossenen WRITE-Auftrag (SINEC-H1) an das **WinCC**-System ('fertig ohne Fehler' (FOF)) muß das DW 0 des Sendefaches mit dem Wert '0' überschrieben werden. Hiermit ist das Sendefach wieder freigegeben, und weitere Meldungsblöcke, sofern vorhanden, können vom internen Ringpuffer in das Sendefach übertragen werden.

Der WRITE-Auftrag (SINEC-H1) ist über die SEND-Direkt-Funktion zu realisieren und ist im entsprechenden Handbuch des AG nachzulesen.

5.2.4 Schnittstellenbeschreibung

Folgende Schnittstellen und Bausteine werden beschrieben:

- System Datenbaustein DB 80:
  Zur Parametrierung des S5 Meldesystems.
- Offset Datenbaustein für die entsprechende Meldungssorte:
  Binäre Schnittstelle der Meldungssignale zum S5-Meldesystem mit Spezifikation der Meldungseigenschaften.
- Parameter Datenbaustein für entsprechende Meldungssorte:
  Zur Angabe zusätzlicher Meldungsdaten der Sorte 2 bis 4.
- Sendefach:
  Übergabeschnittstelle zum **WinCC**-System.

5.2.4.1 System Datenbaustein 80

Mit Hilfe des System-Datenbausteins DB 80 können voneinander unabhängige Datenbereiche für vier Meldungssorten, einem Ringspeicher und einem Sendefach projektiert werden. Für die Projektierung sind Datenwort 0 bis 20 im DB 80 vorgesehen. Eine detaillierte Beschreibung der Datenworte 0 bis 20 ist in Kapitel 5.2.5 gegeben.

5.2.4.2 Offset Datenbaustein

Das S5 Meldesystem wertet die Signalzustände der entsprechenden Meldungen aus und bildet daraus bei Bedarf entsprechende Meldungsblöcke.

Der Anwender hat dafür zu sorgen, daß ...

- bei der Projektierung die Ruhezustände der einzelnen Meldungen angegeben werden.
- die Meldungszustände während der Laufzeit des S5 Anwenderprogramms, in die entsprechenden Signalzustandsbits eingeschrieben werden.
- bei Bedarf die entsprechenden Quittierungsbits ausgelesen und ausgewertert werden.

5.2.4.3 Parameter Datenbaustein

Bei der Meldungssorte 2 bis 4 können über den Meldungsblock zusätzliche Informationen über den aktuellen Anlagenzustand übergeben werden.

Der Anwender hat dafür zu sorgen, daß ...

- bei Eintreffen einer Meldung die gültigen Prozeßvariablen (Prozeßwert, Auftrags- und Chargennummer) in den entsprechenden Parameterblöcken stehen.

5.2.4.4 Sendefach / Übergabefach

Das Sendefach wird sobald es Meldungsblöcke enthält mit einem WRITE-Auftrag (SINEC-H1) direkt an das **WinCC**-System übermittelt (siehe Kapitel 5.2.3.20).

Der Anwender hat dafür zu sorgen, daß ...

- die entsprechenden Hantierungsbausteine der jeweiligen CPU vorhanden sind.
- bei der Projektierung des **WinCC**-Systems entsprechende Kommunikationskanäle für eine Prozeßbus-Anbindung angegeben sind.
- ein WRITE Auftrag, wie in Kapitel 5.2.3.20 beschrieben, angestoßen wird.

5.2.5 Parametrierung des S5 Meldesystems / System DB 80

Beschreibung der projektierbaren Datenwörter des System Datenbausteins DB 80:

DW	Beschreibung
0	DB-Adresse: interner FIFO-Anfang
1	DB-Adresse: interner FIFO-Ende
2	0: ohne Datum und Uhrzeit 1: mit Datum und Uhrzeit
3	DB-Offset für Meldungen der Sorte 1
4	1: einen DB-Offset der Sorte 1 2: zwei DB-Offset der Sorte 1
5	DB-Offset für Meldungen der Sorte 2
6	DB-Offset für Meldungen der Sorte 3
7	DB-Offset für Meldungen der Sorte 4
8	Reserve
9	Reserve
10	DB-Adresse: Sendefach CPU -> PC
11	1: Erfassungsoptimiert (EFOP)
12	EFOP ab n Meldungen
13	AG-Typ (115 / 135 / 155)
14	Reserve (muß 1 sein)
15	AG.Nr.: 1..255; CPU-Nr.: 1..4
16	Reserve
17	Reserve
18	Reserve
19	Reserve
20	PAFE der Plausibilitätsprüfung

Tabelle 14

DW 0, DW 1 : DB-Speicherbereich des internen Ringpuffers

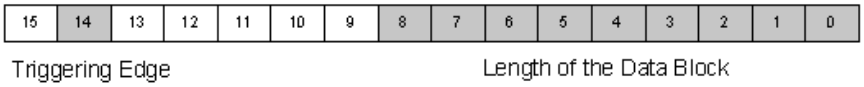
Über die beiden Datenwörtern wird der interne Ringpufferbereich für Meldungen festgelegt.
Der Speicherplatz muß mindestens zwei Datenbausteine groß sein und es ist darauf zu achten, daß das FIFO-Ende größer als der FIFO-Anfang parametriert wird.
Der Speicherbereich des Pufferspeichers ergibt sich aus dem Datenbausteinbereich, welcher durch FIFO-Anfang und FIFO-Ende eingegrenzt wird, einschließlich der beiden angegebenen Datenbausteine.

Die Wahl der Ringpuffergröße:

Bei erreichen der Speicherkapazität des Ringpuffers werden die zeitlich ältesten Meldungen überschrieben. Die DB-Anzahl muß so groß gewählt werden, daß bei einem auftretenden Meldungsschwall keine Meldungen, bevor sie ausgelagert werden konnten, überschrieben werden. Damit dies nicht zum Tragen kommt, gilt folgende Faustformel.

Bestimmung der DB-Anzahl des Ringpuffers:

Meldungen pro DB = (255 DW / DB) / Meldungsblocklänge
siehe Tabelle 10



Bei *Erfassungsoptimierten Betrieb* des Meldungserfassungssystems ist es ratsam ein bis zwei Datenbausteine mehr einzukalkulieren.

DW 2 : Datum- und Uhrzeitkennung

Die Wahl, Meldungen mit einem Datum- und Uhrzeitstempel zu versehen, bezieht sich auf alle parametrierten Meldungen. Entweder erhalten alle zu erfassenden Meldungen einen Datum- und Uhrzeit- Stempel (DW2 = 1) oder keine Meldungen (DW2 = 0). Wird DW2 = 0 gesetzt erweitert das **WinCC**-System die eintreffenden Meldeblöcke mit einem Datum/Uhrzeit Stempel.

DW 3, DW 4 : Offset-DB der Meldungssorte 1

Sind Meldungen der Sorte 1 (Meldungen ohne Parameter und Chargenbezeichnung) zu projektieren, dann ist in Datenwort 3 die Adresse des Offset Datenbausteins anzugeben. In den hier angegebenen Datenbausteinen müssen die Signalzustände dieser Meldungen fortlaufend vom Steuerungsprogramm eingeschrieben werden.

Sind mehr als 1008 Meldungen (maximal 2016 Meldungen) der Sorte 1 vorgesehen, wird durch eintragen der Zahl '2' im Datenwort 4 ein weiterer Datenbaustein für Meldungen der Sorte 1 freigegeben. Der zweite Datenbaustein hat automatisch, bezogen auf die Adresse in DW 3, die nächst höhere Adresse. Bei maximal 1008 Meldungen der Sorte 1 wird in das DW 4 eine '1' eingetragen.

DW 5, DW 6, DW 7 : Offset-DB der Meldungssorte 2, 3, 4

Die Datenwörter 5-7 enthalten analog zu DW 3 die jeweiligen Datenbaustein-Adressen, in dem die Signale der Meldungen abgelegt sind.

In DW 5 steht die Adresse des Datenbausteins für die Meldungssorte 2, in DW 6 entsprechend für die Meldungssorte 3 und in DW 7 für die Meldungssorte 4.

Kommt eine Meldungssorte nicht zum Einsatz, muß in dem entsprechenden DW eine '0' stehen.

Bei den angegebenen Adressen in DW 5-7 handelt es sich um sogenannte 'Offset-DBs'. Je nach Meldungssorte und Meldungszahl pro Sorte sind diesen eine entsprechende Anzahl von 'Folge-DB' zugeordnet. Diese enthalten die Parameter der Meldungen. Deshalb ist bei der Vergabe der Offset-DB-Adressen darauf zu achten, daß zwischen den vorhergehenden Offset-DB und dem anzugebenden genug Platz (Datenbausteine) für die Parameter-DB vorgesehen wird.

Je Meldungsssorte 2-4 sind max. 1008 Meldungen projektierbar. Bei einer vollen Ausnutzung ergibt sich bei unterschiedlichen Sorten folglich eine andere Anzahl von 'Folge-DB' (Parameter-DB) zu den Offset-DB (siehe Tabelle 8).

DW 10 : Meldeschnittstelle zum überlagerten WinCC-System

Das Datenwort 10 muß immer parametriert werden, wobei es ohne Belang ist, in welcher Betriebsart das Meldeerfassungssystem arbeiten soll. In DW 10 wird die DB Adresse des Übergabefaches vergeben. Das Übergabefach dient als Schnittstelle von der SIMATIC S5 zum überlagerten **WinCC**-System.

DW 11 , DW 12 : Betriebswahl für Erfassungsoptimum und entsprechender Meldungsanzahl

Es sind zwei mögliche Betriebsarten vorgesehen:

- '0' in DW 11 -> **'Normalbetrieb'** des Meldungserfassungssystems
- '1' in DW 11 -> **'Erfassungsoptimum - Betrieb'** des Meldungserfassungssystems

Normalbetrieb:

Es werden soviele erfaßte Meldungen innerhalb eines Zyklus aus dem internen Puffer zum Senden ausgelagert, wie die Meldeschnittstelle aufnehmen kann, unter der Voraussetzung, daß sie bereit ist Daten aufzunehmen.
Bei einem sehr großen Meldungsaufkommen innerhalb eines Zyklus bzw. mehrerer aufeinanderfolgender Zyklen würde dieser Ablauf zu einer relativ hohen Zykluszeit führen. Diese wird umso größer, je größer die Meldungsblöcke der beteiligten Meldungssorten sind. Hierbei ist die Erfassung der Meldungsblöcke aufwendiger und länger.

Erfassungsoptimiert:

Bei den auftretenden Meldungen hat die zeitlich chronologische Erfassung vor der Sendung an den PC Priorität. Die relative Zeit zwischen den auftretenden Meldungen der Anlage untereinander steht im Vordergrund. Ob die Meldungen ein paar Millisekunden später bei dem PC ankommen oder nicht ist zweitrangig. Die Trägheit des menschlichen Auges und die Aufnahmefähigkeit des Beobachters in der Warte sind dafür ausschlaggebend.

Um bei solchen zeitkritischen Fällen die Zykluszeit des Meldungserfassungssystems herabsetzen zu können, ist die Betriebsmöglichkeit dieses Systems 'erfassungsoptimiert zu arbeiten' eingeführt worden. Die Mindestanzahl auftretender Meldungen innerhalb eines OB1 Zyklus ist in DW 12 anzugeben. Überschreitet die Meldungsanzahl diese Mindestanzahl während des aktuellen OB1 Zyklus werden die Meldungen nur erfaßt und gepuffert. Auf die Auslagerung und die anschließende Sendung an einen Kopplungspartner wird in diesem OB1 Zyklus verzichtet.

DW 15 : AG- / CPU-Nummer

Dieses Datenwort wird für die Bildung des Telegrammkopfes benötigt und bedarf der Angabe der projektbedingten AG- sowie der CPU-Nummer dieses Automatisierungsgerätes. Die CPU-Nummer ist vor allem dann von Bedeutung, wenn innerhalb eines AG's mehrere CPU's arbeiten. Nur im Zusammenhang mit dem Datenwort, welches die Kennung für Meldungen enthält, kann das überlagerte WinCC-System die gesendeten Daten als Meldung interpretieren, die meldungsspezifischen Meldetexte zuweisen und entsprechend auswerten.

Das DW 15 hat bei der Parametrierung als einziges Datenwort das S5-Datenformat 'KY'. Damit lassen sich zwei Bytes getrennt (durch ein Komma) darstellen. Das linke Byte beinhaltet die AG-Nummer, welche im Bereich 1 bis 255 liegen kann. Im rechten Byte wird die CPU-Nummer, die die Zahl 1 bis 4 betragen darf, angegeben.

Beispiel:

KY = 10,2
AG-Nummer = 10
CPU-Nummer = 2

DW 20 : Parametrierfehler

Alle im System-DB parametrierten Datenwörter werden beim Anlauf des S5-Meldesystems auf deren Plausibilität geprüft. Dabei wird jeweils zwischen Überschreitungen möglicher Wertebereiche, Überschneidungen bzw. Mehrfachbelegung von parametrierten Datenbausteinen und fehlenden Angaben unterschieden.

Als Ausgabeparameter in dem Format eines Datenwortes hat dieser Funktionsbaustein ein sog. PAFE-Wort (Parametrier-FEhler-Wort); dies ist ähnlich den systemspezifischen Hantierungsbausteinen . Der Status des PAFE-Wortes kann im System-DB 80 aus DW 20 entnommen werden. Das PAFE-Wort kann nach dem Programmrücksprung aus dem FB 81 auf aufgetretene Fehler untersucht werden. Im Anschluß daran können entsprechende Handlungen vorgenommen werden.

Sinnvoll ist es, das Automatisierungsgerät bei einem PAFE-Wort, welches von Null verschieden ist, in den Stop-Zustand 'springen' zu lassen. Wird das PAFE-Wort unberücksichtigt gelassen, kann keine Garantie dafür gegeben werden, ob sich das Programm fehlerfrei verhält.

Die Auswertung des PAFE-Wortes
Wird das Programm bzw. AG nach Empfehlung bei auftreten eines Fehlers (PAFE-Wort <> 0) in dessen Stop-Zustand gebracht, kann an Hand der Fehlernummer der Fehler gezielt analysiert und behoben werden. Nachfolgende Tabelle gibt über die Fehlerart, welche bei der Parametrierung verursacht wurde, Auskunft.

Format des PAFE-Wortes:

KY = Fehlernummer, Sammelfehlerkennung

Beispiel:

KY = 9,1

Aufgetretener Parametrierfehler mit der Nummer 9 entspricht: Offset-DB-Adr. d. Sorte 1 ist größer als die max. zulässige DB-Adresse.

Fehlernummern und ihre Bedeutung:

Fehler-Nr	Bedeutung
1	Anfang-DB des int. Puffers ist nicht definiert
2	Anfang-DB des int. Puffers ist Adressengleich mit dem System-DB (’80’)
3	Anfang-DB-Adr. des int. Puffers ist größer als die max. zul. DB-Adresse
4	Ende-DB des int. Puffers ist Adressengleich mit dem System-DB ('80')
5	Ende-DB-Adr. ist kleiner als Anfang-DB-Adr. des int. Puffers
6	Ende-DB-Adr. des int. Puffers ist größer als die max. zul. DB-Adresse
7	Offset-DB d. Sorte 1 ist Adressengleich mit dem System-DB ('80')
8	Offset-DB-Adr. d. Sorte 1 liegt innerhalb des int. Pufferbereichs
9	Offset-DB-Adr. d. Sorte 1 ist größer als die max. zulässige DB-Adresse
10	Offset-DB der Sorte 2 ist Adressengleich mit dem System-DB ('80')
11	Offset-DB d. Sorte 2 ist Adressengleich mit dem der Sorte 1
12	Offset-DB der Sorte 2 ist Adressengleich mit dem 2. Offset-DB d. Sorte 1
13	Offset-DB-Adr. d. Sorte 2 liegt innerhalb des int. Pufferbereichs
14	Offset-DB-Adr. d. Sorte 2 ist größer als die max. zulässige DB-Adresse
15	Offset-DB d. Sorte 3 ist Adressengleich mit dem System-DB ('80')
16	Offset-DB d. Sorte 3 ist Adressengleich mit dem der Sorte 1
17	Offset-DB d. Sorte 3 ist Adressengleich mit dem 2. Offset-DB der Sorte 1
18	Offset-DB d. Sorte 3 ist Adressengleich mit dem der Sorte 2
19	Offset-DB-Adr. d. Sorte 3 liegt innerhalb des int. Pufferbereichs
20	Offset-DB-Adr. d. Sorte 3 ist größer als max. zulässige DB-Adresse
21	Offset-DB d. Sorte 4 ist Adressengleich mit dem System-DB ('80')
22	Offset-DB d. Sorte 4 ist Adressengleich mit dem der Sorte 1
23	Offset-DB d. Sorte 4 ist Adressengleich mit dem 2. Offset-DB der Sorte 1
24	Offset-DB-Adr. d. Sorte 4 liegt innerhalb des int. Pufferbereichs
25	Offset-DB-Adr. d. Sorte 4 ist größer als die max. zulässige DB-Adresse
26	Offset-DB d. Sorte 4 ist Adressengleich mit dem der Sorte 2
27	Offset-DB d. Sorte 4 ist Adressengleich mit dem der Sorte 3
28	PC-Sendefach ist Adressengleich mit dem System-DB ('80')
29	PC-Sendefach ist nicht definiert ('0')
30	PC-Sendefach-Adr. liegt innerhalb des int. Pufferbereichs
31	PC-Sendefach-Adr. ist größer als die max. zulässige DB-Adresse
32	PC-Sendefach ist Adressengleich mit dem Offset-DB der Sorte 1
33	PC-Sendefach ist Adressengleich mit dem Offset-DB der Sorte 2
34	PC-Sendefach ist Adressengleich mit dem Offset-DB der Sorte 3
35	PC-Sendefach ist Adressengleich mit dem Offset-DB der Sorte 4
36	PC-Sendefach ist Adressengleich mit dem 2. Offset-DB der Sorte 1
37	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
38	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
39	Reserve DW 9 bzw. Reserve DW 10 ungleich 0

Fehler-Nr	Bedeutung
40	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
41	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
42	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
43	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
44	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
45	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
46	Reserve DW 9 bzw. Reserve DW 10 ungleich 0
47	Meldungsanzahl f. d. Mindestgrenze der gewählten Betriebsart mit Erfassungsoptimum fehlt
48	AG-Typ ist nicht definiert
49	Reserve DW 14 ungleich 1
50	AG-Nr. für Telegrammkopf ist nicht definiert
51	CPU-Nr. für Telegrammkopf ist nicht definiert
52	CPU-Nr. ist größer als zulässig (1..4)

Tabelle 15

5.2.6 Projektierungsbeispiel für das S5 Meldesystem

Beschreibung

Das S5 Meldesystem soll für die folgenden Meldungssorten projektiert werden:

Sorte	Definition: Meldesorten
1	1200 Meldungen (von Meldungsnummer 10000 bis 11199) Meldungen 11000 bis 11199 sind 'low aktiv'
2	keine Meldungen vorgesehen
3	11 Meldungen (von Meldungsnummer 30000 bis 30010)
4	keine Meldungen vorgesehen

Alle Meldungen erhalten einen Datum/Uhrzeitstempel.
Verwendet wird eine 135U, AG Nummer 1, CPU Nummer 1.

5.2.6.1 Parametrierung DB 80

Sorte	max. Anzahl	Größe des Parameter-Blocks	Anzahl Blöcke je Parameter DB	max. Anzahl Parameter DBs
1	1008 / 2016	-	-	-
2	1008	2 DW	128	8
3	1008	5 DW	51	20
4	1008	7 DW	36	28

DB 81 wird als PC - Sendefach benutzt.

Bei einem gleichmäßigen Auftreten der vorhandenen Meldungen ergibt sich eine mittlere Meldungsblocklänge (mit Datum/Uhrzeit) von:
(1200 * 5 + 11 * 10) / (1200 + 11) = 5,04

Annahme:

Das S5-Meldesystem soll einen Meldeschwall von 100 Meldungen in einem AG-Zyklus aufnehmen können und ab 30 Meldungen im 'Erfassungsoptimierten Betrieb' arbeiten.

5 DW/Mld. * 100 Mld = 500 DW
(500 DW) / (256 DW/DB) = 1,95 DBs

Für den Ringpuffer ergeben sich somit vier Datenbausteine, da für den 'Erfassungsoptimierten Betrieb' ein bis zwei Datenbausteine mehr gerechnet werden sollten. Der Ringpuffer beginnt bei Datenbausteinadresse 82 und somit ergibt sich eine Endadresse das Ringpuffers von DB 85.

Um eine Reserve für einen zukünftigen Ausbau des Ringpuffers vorzusehen, liegt der Offset-Datenbaustein der Sorte 1 auf DB 88 und DB 89 (mehr als 1008 ´Meldungen der Sorte 1).

DB 90 wird zum Offset-Datenbaustein der Meldungssorte 3. Ein Parameter-DB der Meldungssorte 3 hat ein Aufnahmevolumen von 51 Parameter-Blöcken, subtrahiert man die benutzten 11 Blöcke ergibt sich mit nur einem Parameterbaustein (DB 91) eine Erweiterbarkeit von 40 Meldungen der Sorte 3.

DW	Beschreibung	Wert
0	DB-Adresse: interner FIFO-Anfang	82
1	DB-Adresse: interner FIFO-Ende	85
2	0: ohne Datum und Uhrzeit 1: mit Datum und Uhrzeit	1
3	DB-Offset für Meldungen der Sorte 1	88
4	1: einen DB-Offset der Sorte 1 2: zwei DB-Offset der Sorte 1	2
5	DB-Offset für Meldungen der Sorte 2	0
6	DB-Offset für Meldungen der Sorte 3	90
7	DB-Offset für Meldungen der Sorte 4	0
8	Reserve	0
9	Reserve	0
10	DB-Adresse: Sendefach CPU -> PC	81
11	1: Erfassungsoptimiert (EFOP)	1
12	EFOP ab n Meldungen	30
13	AG-Typ (115 / 135 / 155)	135
14	Reserve	1
15	AG.Nr.: 1..255; CPU-Nr.: 1..4	1, 1
16	Reserve	0
17	Reserve	0
18	Reserve	0
19	Reserve	0
20	PAFE der Plausibilitätsprüfung	0

Datenbaustein 100 wird von DW 10 bis DW 20 zur Uhrzeitsynchronisierung verwendet.
Datenbaustein 101 wird von DW 0 bis DW 255 für die Aufnahme von Kommandos verwendet.

5.2.6.2 Einrichten der Datenbausteine

Anlegen der Datenbausteine DB 81 - DB 85, DB 88 - DB 91 und DB 101 von DW 0 - DW 255.
Anlegen des Datenbausteins DB 100 von DW 0 - DW 20.

5.2.6.3 Initialisierung der Offset Datenbausteine

Meldungssorte 1

Für die Meldesorte 1 sind DB 88 und DB 89 vorgesehen. DB 88 beinhaltet die Meldungen mit den Meldenummern 10000 bis 11007, DB 89 die Meldungen mit den Meldenummern 11008 bis 11199.
Insgesamt sollen 1200 Meldungen der Sorte 1 projektiert werden.

Siehe Kapitel 5.2 Adresse des letzten Signalzustandsblocks :
Offsetmeldungsnummer = Meldungsnummer - Basismeldungsnummer = 0 bis 1199

1. Offset DB:

Adresse des letzten Signalzustandsblocks: DW 252

2. Offset DB:

Adresse des letzten Signalzustandsblocks: DW 252

1200 - 1008 = 192

192 / 16 = 12
192 % 16 = 0

Adresse des letzten
Signalzustandblock's im Offset Datenbaustein 2 = 12 * 4 = 48

DB 88:

DW	Beschreibung	Wert
DW 0	frei	
DW 1	Basismeldungsnummer	10000
DW 2	Adresse des letzten DW	252
DW 3	frei	

DB 89:

DW	Beschreibung	Wert
DW 0	frei	
DW 1	Basismeldungsnummer	11018
DW 2	Adresse des letzten DW	48
DW 3	frei	

Siehe Kapitel 5.2.3.3 Offsetmeldungsnummer/Signalzustände der Meldung.

Die Meldungen 11000 bis 11199 sind 'low aktiv'.

Position des Ruhezustandsbits der Meldungsnummer 11000:

Offsetmeldungsnummer:	11000 - 10000 = 1000
Anfang des Signalzustandsblocks	(Offsetmeldungsnummer / 16 + 1) * 4 =
	= (62 + 1) * 4 * DW 252
Datenwort der Ruhezustandsbits:	DW 253
Datenbit:	Offsetmeldungsnummer % 16 = 8
Datenbaustein:	Offset Datenbaustein = DB 88

Position des Ruhezustandsbits der Meldungsnummer 11000:

Offsetmeldungsnummer:	11199 - 10000 = 1199
Anfang des Signalzustandsblocks:	(Offsetmeldungsnummer / 16+ 1) * 4 =
	= (74 +1) * 4 = 300
	300 - 252 = 48
Datenwort der Ruhezustandsbits:	DW 49
Datenbit:	Offsetmeldungsnummer % 16 = 15
Datenbaustein:	Offset Datenbaustein + 1 = DB 89

Folgende Ruhezustandsbits sind zu ändern:

DB 88:

DW 253: Datenbit 8 bis 15 auf '1' setzen

DB 89:

DW 5, DW 9, DW 13, ... DW 49 : Datenbit 0 bis 15 auf '1' setzen

Meldungssorte 3

Für die Meldesorte 3 ist DB 90 als Offset Datenbaustein mit den Meldungen 30000 bis 30010 und DB 91 als Parameter Datenbaustein vorgesehen.
Insgesamt sollen 11 Meldungen der Sorte 3 projektiert werden.
Siehe Kapitel 5.2.3.10 Aufbau des Parameter Datenbausteins.

OffsetmeldungsNr. = MeldungsNr. - BasismeldungsNr. = 0 bis 10

Offset DB:

Adresse des letzten Signalzustandsblocks:		11 / 16 = 0
		11 % 16 = 11
Adresse des letzten Signalzustandsblocks	=	(0+1) * 4 = 4

DB 89:

DW	Beschreibung	Wert
DW 0	frei	
DW 1	Basismeldungsnummer	30000
DW 2	Adresse des letzten DW	4
DW 3	frei	

Alle Ruhezustandsbits sind '0'.
Siehe Kapitel 5.2.3.10 Aufbau des Parameterdatenbausteins

Parameter DB = Offset DB + 1 + (Offsetmeldungsnummer / Parameterblöcke je Parameter DB)

Meldungsnummer 30000:

Parameter DB = 90 + 1 + 0 / 51 = 91

Meldungsnummer 30010:

Parameter DB = 90 + 1 + 10 / 51 = 91

Anfangsadresse des jeweiligen Parameterblocks = (Offsetmeldungsnummer % Parameterblöcke je Parameter DB) * Größe des Parameterblock

Meldungsnummer 30000:

Anfangsadresse des jeweiligen Parameterblocks = (0 % 51) * 5 = DW

Meldungsnummer 30010:

Anfangsadresse des jeweiligen Parameterblocks = (10 % 51) * 5 = DW 50

DB 91: Parameter Datenbaustein 91 zu Offset Datenbaustein 90

MeldungsNr.	Prozeßwerte	Auftragsnummer	Chargenbezeichnung
30000	DW 0, 1	DW 2, 3	DW 4
30001	DW 5, 6	DW 7, 8	DW 9
30002	DW 10, 11	DW 12, 13	DW 14
30003	DW 15, 16	DW 17, 18	DW 19
30004	DW 20, 21	DW 22, 23	DW 24
30005	DW 25, 26	DW 27, 28	DW 29
30006	DW 30, 31	DW 32, 33	DW 34
30007	DW 35, 36	DW 37, 38	DW 39
30008	DW 40, 41	DW 42, 43	DW 44
30009	DW 45, 46	DW 47, 48	DW 49
30010	DW 50, 51	DW 52, 53	DW 54

5.2.7 Dokumentation der SIMATIC S5 Kommandobausteine

Aufgabe und Funktion der S5-Kommandobausteine

Die Software dient zum 'Bearbeiten' von Bits, Bytes, Worten und Doppelworten in der SIMATIC S5 über einen Prozeßbus (z.B. SINEC H1). Über den Prozeßbus ist es nur möglich Byte- bzw. Wortwerte in der SIMATIC S5 anzusprechen.
Folgende Operationen sind standardmäßig ausführbar:

- Datenbausteine (DB und DX), Timer und Counter sind nur als Wort zu ändern.
- Merker, Eingänge, Ausgänge, Peripherie (P und Q) sind nur als Byte zu ändern.

Das Programmpaket stellt innerhalb der SIMATIC S5 die notwendige Softwarefunktionalität zur Verfügung, um die folgende Operationen vom WinCC-System aus über den gegebenen Prozeßbus zu realisieren:

- Für einen OB1-Zyklus lang einen Richtimpuls setzten
- Bit in DB / DX setzen / rücksetzen / invertieren
- Bit in Merker setzen / rücksetzen / invertieren
- Byte links / rechts in DB / DX schreiben
- Wort / Doppelwort in DB / DX schreiben
- Byte / Wort in Merker schreiben
- Byte / Wort in Peripherie schreiben
- Byte / Wort in erweiterte Peripherie schreiben

Die gewünschten Änderungen in der SIMATIC S5 werden über eine Daten-Schnittstelle vom WinCC - Control Center als Rohdatenvariable zur Verfügung gestellt. Die Befehle müssen über die Rohdatenvariable an die S5 geschickt werden. Diese Befehle werden direkt in der S5 durch einen Befehlsinterpreter **FB 87 : EXECUTE** ausgewertet und ausgeführt.

Das vorliegende Handbuch beschreibt detailliert den Einsatz und die Handhabung der S5-Kommandobausteine in der SIMATIC S5 - Umgebung. Der Anwender erhält eine Übersicht über die von der Software verwendeten Funktions- und Datenbausteine, sowie den benötigten Speicherplatz. Es folgt eine detaillierte Schnittstellenbeschreibung der vorhandenen Datenschnittstelle. Als Hilfestellung ist ein Projektierungsbeispiel aufgeführt.

5.2.7.1 Auflistung der Softwarebausteine

Die SIMATIC S5 Software *'S5-Kommandobausteine'* befindet sich auf der WinCC - CD unter dem Dateinamen **WINCC1ST.S5D**.

Die Datei beinhaltet folgende Funktionsbausteine für die 'S5-Kommandobausteine':

FB	Name	Größe in Byte	Funktion
FB 87	EXECUTE	152	Ermöglicht Bit-, Byte-, Wort-, Doppelwort - Manipulationen über den Prozeßbus
FB 88	OPCODE	399	Aufgerufen durch FB 87
Gesamt		551	

Tabelle 16

Zusätzllich wird ein Kommando Datenbaustein mit der Größe 512 Byte benötigt.

5.2.7.2 Hardwarevoraussetzung

Die in der Tabelle 16 angegebenen Funktionsbausteine benötigen zur korrekten Ausführung folgende Hardware:

AG	CPU
AG 115U	CPU 943, CPU 944, CPU 945
AG 135U	CPU 928A, CPU 928B
AG 155U	CPU 946/ 947, CPU 948

5.2.7.3 Aufrufparameter von FB 87: EXECUTE

Im folgenden werden die Aufrufparameter von Funktionsbaustein **FB 87 : EXECUTE** beschrieben.

Name	Execute	Parameter
Bez:	DBNR	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
Bez:	DBDX	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
Bez:	RIMP	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY

DBNR: Datenbaustein-Nummer der Kommandoübergabe-Schnittstelle

DBDX: Typ der Datenquelle für die Kommandoübergabe-Schnittstelle

DB:Datenquelle ist ein Datenbaustein (DB).
DX:Datenquelle ist ein erweiterter Datenbaustein (DX).

RIMP: Bitposition für den Richtimpuls

RIMP:Merker-Nummer , Bit-Nummer

5.2.8 Schnittstellenbeschreibung

Folgende Schnittstellen und Bausteine werden beschrieben:

- Komando-Funktionsbaustein FB 87
- Kommando-Datenbaustein: Befehlsübergabeschnittstelle zur SIMATIC S5.

In der SIMATIC S5 wird im OB 1 der Befehlsinterpreter (**FB 87 : EXECUTE**) zyklisch aufgerufen. Art und Adresse des Kommando DB's werden als Parameter übergeben. Bei anstehendem Kommando wird der Opcode und vier Parameter an **FB 88 : OPCODE** weitergeleitet und direkt ausgeführt. Nach ausgeführtem Befehl wird der Kommandozähler (DW 1) um eins dekrementiert. Der Vorgang der Befehlsübergabe und des Dekrementierens des Kommandozählers wiederholt sich bis alle anstehenden Befehle abgearbeitet wurden.

Angaben über Typ und Adresse des Datenbausteins müssen im WinCC Control-Center, als auch im S5 Programm übereinstimmen und der Datenbaustein in der S5 vorhanden sein. Als Auswahl steht ein DB- bzw. ein DX-Datenbaustein und dessen Adresse (z.B. DX 234) zur Verfügung. Der Datenbaustein muß bis Datenwort 255 vom Anwender geöffnet werden, da Datenwort 0-255 im angegebenen Datenbaustein angesprochen werden können.

Folgende Syntax der abgelegten Kommandos im Kommando-Datenbaustein ist festgelegt:

DW	Beschreibung
0	nicht benutzt
1	Anzahl der auszuführenden Kommandos
2	Opcode des ersten Kommandos
3	Parameter 1 (Opcode 1)
4	Parameter 2 (Opcode 1)
5	Parameter 3 (Opcode 1)
6	Parameter 4 (Opcode 1)
7	Opcode des zweiten Kommandos
8	Parameter 1 (Opcode 2)
9	Parameter 1 (Opcode 2)
10	Parameter 2 (Opcode 2)
11	Parameter 3 (Opcode 2)
12	Parameter 4 (Opcode 2)
13	Opcode des dritten Kommandos
14	Parameter 1 (Opcode 3)
.....	

Nachfolgend sind die Implementierten Kommandos syntaktisch beschrieben:

Übergabe von Opcode und Parametern in den Kommando DB

Befehl	Opode	Parameter 1	Parameter 2	Parameter 3	Parameter 4
Bit in DB setzen	10	DB	DW	Bit	-
Bit in DB rücksetzen	11	DB	DW	Bit	-
Bit in DB invertieren	12	DB	DW	Bit	-
Byte rechts in DB setzen	15	DB	DW	Wert	-
Byte links in DB setzen	16	DB	DW	Wert	-
Datenwort in DB schreiben	17	DB	DW	Wert	-
Doppelwort in DB schreiben	18	DB	DW	Wert	Wert
Bit in DX setzen	20	DX	DW	Bit	-
Bit in DX rücksetzen	21	DX	DW	Bit	-
Bit in DX invertieren	22	DX	DW	Bit	-
Byte rechts in DX setzen	25	DX	DW	Wert	-
Byte links in DX setzen	26	DX	DW	Wert	-
Datenwort in DX schreiben	27	DX	DW	Wert	-
Doppelwort in DX schreiben	28	DX	DW	Wert	Wert
Merkerbit setzen	30	MB	Bit	-	-
Merkerbit rücksetzen	31	MB	Bit	-	-
Merkerbit invertieren	32	MB	Bit	-	-
Merkerbyte schreiben	35	MB	Wert	-	-
Merkerwort schreiben	36	MW	Wert		-
Peripheriebyte schreiben	45	PB	Wert	-	-
Peripheriewort schreiben	46	PW	Wert	-	-
Erweitertes Peripheriebyte schreiben	55	QB	Wert	-	-
Erweitertes Peripheriewort schreiben	56	QW	Wert	-	-
Richtimpuls setzen	60	-	-	-	-

5.2.8.1 Projektierungsbeispiel für die S5 Kommandobausteine

Die S5 Kommandobausteine sollen eingerichtet werden.
Der Richtimpuls wird in Merkerwort 56, Bit 4 zur Verfügung gestellt. Als Kommando-Datenbaustein soll DX 237 dienen. Es ist dafür zu sorgen, daß im AG der Datenbaustein DX 237 von DW 0 bis 255 geöffnet ist.
Im WinCC Control-Center ist bei der Angabe der Kanalparameter (z.B. SINEC H1) der gewünschte Datenbaustein einzutragen.

5.2.9 Aufgabe und Funktion der S5-Uhrzeitsynchronisation

Die vorliegende Dokumentation beschreibt die Funktionen und Eigenschaften der SIMATIC S5 Software:

S5-Uhrzeitsynchronisation
Die Software dient zur Synchronisation der SIMATIC S5 Systemuhr. Weiterhin liefert sie der 'zeitfolgerichtigen Meldungserfassung' des S5-Meldesystems ein passendes Datum-/Uhrzeit-Datenformat zur Bildung der Meldungsblöcke.

Der Funktionsbaustein **FB 86 : MELD:UHR** stellt zusätzlich die aktuelle S5-Uhrzeit in einem für die 'zeitfolgerichtige Meldungserfassung' benötigten Format zur Verfügung. Die Daten liegen im System Datenbaustein 80 ab DW 190 zur Verfügung.

Tritt ein Wechsel in einem Meldungs-Signalzustand auf, so wird die Meldung mit Hilfe der Meldungsnummer vom Funktionsbaustein **FB 80 : SYSTEMFB** identifiziert und mit dem aktuellen Datum- /Uhrzeit-Stempel aus System-Datenbaustein 80 versehen.

Das vorliegende Handbuch beschreibt detailliert den Einsatz und die Handhabung der S5-Uhrzeitsynchronisation in der SIMATIC S5-Umgebung. Der Anwender erhält eine Übersicht über die von der Software verwendeten Funktions- und Datenbausteine, sowie den benötigten Speicherplatz. Als Hilfestellung ist ein Projektierungsbeispiel aufgeführt.

5.2.9.1 Auflistung der Softwarebausteine

Die SIMATIC S5 Software (S5-Uhrzeitsynchronisation) befindet sich auf der WinCC-CD unter dem Dateinamen: WINCC1ST.S5D.

Die Datei beinhaltet folgende Funktions- und Datenbausteine:

FB	Name	Größe in Byte	Funktion
FB 86	MELD:UHR	1135	Synchronisation der Uhrzeit
Gesamt		1135	

Tabelle 17

Uhrendatenbereich 115U: 27 DW = 54 Byte
Uhrendatenbereich 135U/155U: 12 DW = 24 Byte

Datenbereich für das S5 Meldesystem: 3 DW = 6 Byte

5.2.9.2 Hardwarevoraussetzung

Die für das S5 Meldesystem angegebenen Funktionsbausteine benötigen zur korrekten Ausführung folgende Hardware:

AG	CPU
AG 115U	CPU 944 * , CPU 945
AG 135U	CPU 928B
AG 155U	CPU 946/ 947, CPU 948

* nur die CPU 944 mit zwei PG-Schnittstellen besitzt eine Systemuhr

5.2.10 Aufrufparameter von FB 86 : MELD:UHR

Im folgenden werden die Aufrufparameter von Funktionsbaustein **FB 86 : MELD:UHR** beschrieben.

Im folgenden werden die Aufrufparameter von Funktionsbaustein **FB 87 : EXECUTE** beschrieben.

Name	MELD:UHR	Parameter
Bez:	CPUT	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
Bez:	DCF7	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
Bez:	QTYP	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
Bez:	QSYN	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY
Bez:	UDAT	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY
Bez:	ZINT	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF
Bez:	ZUHR	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY
Bez:	ZSYN	E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF

CPUT:

Nr. der CPU	Typ
1	CPU 943 / CPU 944
2	CPU 945
3	CPU 928B
4	CPU 946 / 947
5	CPU 948

DCF7:

Betriebsmodus

0 = Betrieb mit S5 - System - Uhr
1 = Betrieb mit DCF77 - Funkuhr

QTYP:

Typ der Datenquelle für das Uhrzeit-Synchronisations-Telegramm.

0 = Datenquelle ist ein Datenbaustein (DB)
1 = Datenquelle ist ein erweiterter Datenbaustein (DX)

QSYN:

Datenquelle der Uhrzeitdaten

DCF7 = 0: QSYN = DB-Nummer, DW-Nummer des empfangenen Uhrzeitsynchronisationstelegramm
DCF7 = 1: QSYN = DB-Nummer, DW-Nummer der DCF77 - Uhrzeit

UDAT:

Adresse des Uhrendatenbereichs

UDAT = DB-Nummer, DW-Nummer

ZINT:

Zeitintervall in Minuten für das Senden des Synchronisations - Telegramms (DCF7 = 1)

ZUHR:

Zieldatenbereich für Uhrzeitdaten im Meldesystem - Format.

ZUHR = DB-Nummer , DW-Nummer

ZSYN:

Zieldatenbereich für Uhrzeit-Synchronisations-Telegramm(DCF7 = 1)

Soll die Funktionalität der 'zeitfolgerichtigen Meldungserfassung' mit dem S5 Meldesystem eingesetzt werden, wird ein spezielles Uhrzeit-Datenformat in DB 80 ab DW 190 erwartet. Dieses Uhrzeit-Datenformat wird aus der S5 System-Uhrzeit gewonnen und in den entsprechenden Datenbereich ZUHR (DB 80, DW 190-192) geschrieben.

Zusammenhang zwischen ’zeitfolgerichtigem Melden’ und FB 86 : MELD:UHR :

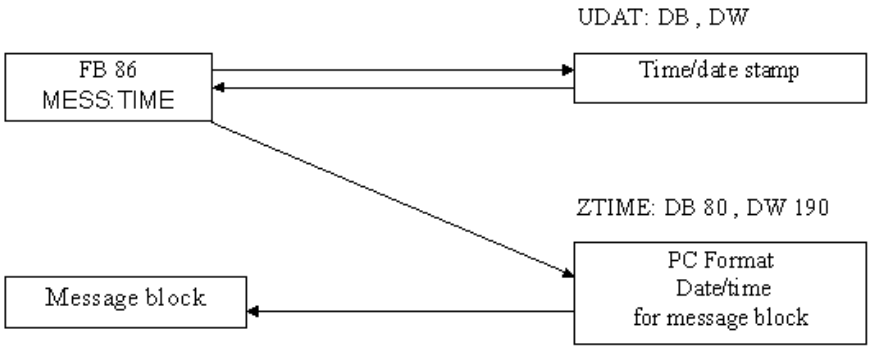


Abbildung 4

5.2.11 Datenformate für Datum und Uhrzeit

Uhrzeit - Synchronisationstelegramm von einem System aus(WinCC unterstützt zur Zeit das Uhrzeit - Telegramm nicht !!)

Im Ersten Datenwort des Uhrzeit-Synchronisations-Telegramm steht eine Quell-Kennung, die vom System mit den Datum- und den Uhrzeitdaten mitgesendet wird.
Der Funktionsbaustein **FB 86 : MELD:UHR** entnimmt das anstehende Telegramm erst, sobald an dieser Stelle die Quell-Kennung 'FFFF' ansteht. Der Empfang des Telegramms wird mit einer '0' in diesem Datenwort quittiert. Erst bei eintreffen eines neuen Telegramms (DW 1 = 'FFFF') wird es erneut ausgelesen und ausgewertet.

Bedeutung	Datenwort	Inhalt	Gültigkeitsbereich	Bemerkung
Quell-Kennung / Uhrzeittelegramm	1	FFFF		
Telegramm - ID	2	FFFF		nicht Benutzt
Sekunden	3	00xx	xx: 0..59	
Minuten	4	00xx	xx: 0..59	
Stunden	5	00xx	xx: 0..23	
Tag	6	00xx	xx: 1..31	
Monat	7	00xx	xx: 1..12	
Jahr	8	00xx	xx: 0..127 (1990-2117)	Jahr + 1990
Tag der Woche	9	00xx	xx: 0..6	Sonntag = 0
Tag des Jahres	10	00xx	xx:1..365	
Sommerzeit, Winterzeit Schaltjahr	11	yyxx	xx: Winterzeit = 00 Sommerzeit = 01 yy: Schaltjahr aktuelles Jahr = 00 letztes Jahr = 01 vor zwei Jahren = 02 vor drei Jahren = 03	

Tabelle 18

5.2.11.1 Uhrendatenbereich CPU 944, CPU 945

Die Datenwortnummern sind relative Angaben. Die tatsächliche Lage des Bereichs wird bestimmt durch die Aufrufparameter : UDAT = DB-Nr., DW-Nr. von **FB 86 : MELD:UHR.**

DW	Location
0	internal tags
1	
2	
3	
4	current time
5	
6	
7	
8	time operating range
9	
10	
11	
12	reserve - alarm time
13	
14	
15	
16	reserve - operating hours
17	
18	
19	
20	
21	
22	time / date after RUN / STOP
23	
24	
25	
26	

Tabelle 19

Aktuelle Uhrzeit im Uhrendatenbereich:

DW	Wort/Links	Wort/Rechts
4	---	Wochentag
6	Tag	Monat
7	Jahr	AM/PM (Bit, Nr.7), Stunde
8	Minute	Sekunde

Abbildung 5

Stellbereich im Uhrendatenbereich:

DW	Wort/Links	Wort/Rechts
9	Schaltjahr	Wochentag
10	Tag	Monat
11	Jahr	AM/PM (Bit, Nr.7), Stunde
12	Minute	Sekunde

Abbildung 6

5.2.11.2 Uhrendatenbereich CPU 928B, CPU 948

Die Datenwortnummern sind relative Angaben. Die tatsächliche Lage des Bereichs wird durch die Aufrufparameter : UDAT = DB-Nr., DW-Nr. von **FB 86 : MELD:UHR** bestimmt.

DW	Location
0	internal tags
1	
2	
3	
4	current time
5	
6	
7	
8	time operating range
9	
10	
11	

Abbildung 7

Aktuelle Uhrzeit im Uhrendatenbereich:

DW	Wort/Links								Wort/Rechts							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4	Sekunden								0							
5	Format		Stunden						Minuten							
6	Monatstag								Wochentag				0			
7	Jahr								Sekunde							

Abbildung 8

Aktuelle Uhrzeit im Stellbereich:

DW	Wort/Links								Wort/Rechts							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8	Sekunden								0							
9	Format		Stunden						Minuten							
10	Monatstag								Wochentag				0			
11	Jahr								Sekunde							

Abbildung 9

5.2.11.3 Uhrendatenbereich CPU 946, CPU 947

Die Datenwortnummern sind relative Angaben. Die tatsächliche Lage des Bereichs wird durch die Aufrufparameter : UDAT = DB-Nr., DW-Nr. von **FB 86 : MELD:UHR** bestimmt.

DW	Location
0	internal tags
1	
2	
3	
4	current time
5	
6	
7	
8	time operating range
9	
10	
11	

Abbildung 10

Aktuelle Uhrzeit im Uhrendatenbereich:

DW	Wort/Links		Wort/Rechts	
4	10er Sek.	1er Sek.	1/10 Sek.	1/100 Sek.
6	10er Std.	1er Std.	10er Min.	1er Min.
7	10er Tag	1er Tag	Wochentag	0
8	10er Jahr	1er Jahr	10er Monat	1er Monat

Abbildung 11

Aktuelle Uhrzeit im Stellbereich:

DW	Wort/Links		Wort/Rechts	
9	10er Sek.	1er Sek.	1/10 Sek.	1/100 Sek.
10	10er Std.	1er Std.	10er Min.	1er Min.
11	10er Tag	1er Tag	Wochentag	0
12	10er Jahr	1er Jahr	10er Monat	1er Monat

Abbildung 12

5.2.11.4 Uhrendatenformat für Meldungsblöcke

Die Datenwortnummern sind relative Angaben. Die tatsächliche Lage des Bereichs wird durch die Parameter : ZUHR = DB-Nr., DW-Nr. von **FB 86 : MELD:UHR** bestimmt. Soll die Funktionalität der 'Zeitfolgerichtigen Meldungserfassung' mit dem S5 Meldesystem verwendet werden, so sind in den Parameter ZUHR die Daten DB 80, DW 190 einzutragen. Das Datum und die Uhrzeit wird von Funktionsbaustein **FB 86 : MELD:UHR**, für die Meldungsverarbeitung im Dual-Code zur Verfügung gestellt:

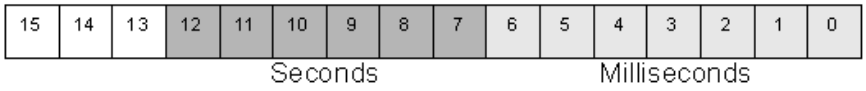
Aktuelle Uhrzeit im Stellbereich:

Bedeutung	Datenwort	Bit	Gültigkeitsbereich	Bemerkung
1/100 Sekunden	1	0 - 6	0..99 (0 - 990 ms)	in einem 10 ms Raster
Sekunden	1	7 - 12	0..59	
Minuten	0	0 - 5	0..59	
Stunden	0	6 - 10	0..23	
Tag	2	0 - 4	1..31	
Monat	2	5 - 8	1..12	
Jahr	2	9 - 15	0..127 (1990-2117)	Jahr + 1990

Abbildung 13

DW3: Uhrzeit

DW4: Uhrzeit



DW4: Datum

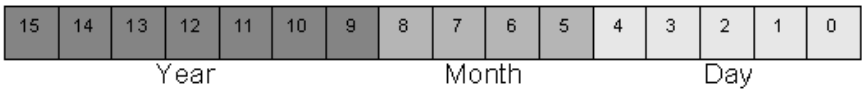


Abbildung 14

5.2.12 Schnittstellenbeschreibung

Um die Software S5 Uhrzeitsynchronisation einzusetzen, muß der Anwender:

- Die Aufrufparameter von **FB 86 : MELD:UHR** entsprechend Kapitel 5.2.10 Aufrufparameter von FB 86 : MELD:UHR ausfüllen
- Die Datenbereiche im AG zu öffnen.

Projektierungsbeispiel

Gegeben ist eine CPU 944 mit zwei PG-Schnittstellen. Auf dieser CPU soll die S5 Uhrzeitsynchronisation für das S5 Meldesystem ohne DCF77-Uhr eingerichtet werden.

Datenbereiche:

Uhrzeit-Synchronisations-Telegramm:	DB 100, DW 20 - DW 30
Uhrendatenbereich der S5 Systemuhr:	DB 100, DW 31 - DW 47
Meldungsblockdaten*:	DB 80, DW 190 -DW 192

* Bei Verwendung des SIMATIC S5 Meldesystems ist dieser Datenbereich fest vorgeschrieben.

Im System ist bei der Angabe der Kanalparameter (z.B. SINEC H1) der gewünschte Datenbaustein (DB 100, DW20 - DW 30) bei der Angabe ‘Uhrzeit-Synchronisation:’ einzutragen.

Es ist dafür zu sorgen, daß DB 80 von DW 0 bis DW 255 und DB 100 von DW 0 bis DW 47 geöffnet sind.

Auszug aus OB 1:

```
: SPA   FB 86
NAME : MELD: UHR
CPUT  : KF +1
DCF7  : KF +0
QTYP  : KF +0
QSYN  : KY 100, 20
UDAT  : KY 100, 31
ZINT   : KF +0
ZUHR  : KY 80, 190
ZSYN  : KF +0
```

5.2.13 Zusammenspiel mit dem WinCC - Meldesystem

Folgendes ist beim Zusammenspiel des WinCC-Meldesystems mit dem S5-Meldebaustein zu beachten :

In der S5 muß im Sendebaustein als Anzahl zu übertragene Datenworte 256 angegeben werden.

Im Control-Center muß im S5Trsp-Kanal eine neue Verbindung eingerichtet werden. Beim Punkt Verbindung/Read-Funktion den Typ Fetch-passiv angeben.

Für den Datenaustausch mit dem Meldesystem müssen pro AG 2 Rohdatenvariablen angelegt werden.
Die erste ist für das Empfangen von Meldungen zuständig.
Ihre Addressierung ist wie folgt einzustellen : Datenbereich : DB, DB-Nr. : xx, Adressierung Wort, DW : 0, Rohdatentyp : Ereignis
Die zweite wird für das Senden der Quittungsinformationen benötigt
Ihre Addressierung ist wie folgt einzustellen : Datenbereich : DB, DB-Nr. : 80, Adressierung Wort, DW : 90, Rohdatentyp : Ereignis

Im Meldesystem die Ereignisvariable mit der Empfangsrohdatenvariable verbinden(Die Bitinformation ist hier ohne Bedeutung).
Die Quittungsvariable mit der Senderohdatenvariable verbinden(Die Bitinformation ist hier ohne Bedeutung).
Als Normierungs-DLL die S5STD.NLL - Datei eintragen.
Tip : Mit dem Verschaltungs-Wizard können alle betroffenen Meldungen auf einmal verbunden werden.

Für die Prozeßwerte sind nur positive Festpunkt-Zahlen zulässig. Gleitkommawerte werden nicht unterstützt.

5.3 Schnittstelle Normierungs-DLL zu Alarm Logging und Tag Logging

Zielsetzung

Die Applikationen AlarmLogging und TagLogging erfassen die Prozeßdaten über den WinCC Datenmanager. Abhängig vom Kommunikationstyp zum Prozeß sind

- unterschiedliche Kanal-DLL's am Datentransfer beteiligt,
- die Prozeßdaten in Telegrammen (Rohdatenvariablen) mit unterschiedlichem Aufbau abgelegt.

AlarmLogging und TagLogging sollen die Prozeßdaten jedoch unabhängig vom jeweiligen Kommunikationstyp auf dieselbe Art und Weise verarbeiten können. Deshalb wird für jeden Kommunikationstyp eine eigene Normierungs-DLL eingesetzt, die den genauen Aufbau der jeweiligen Telegramme kennt und daraus eine für AlarmLogging und TagLogging allgemein gültige „Prozeßdatenform" ableitet.

Eine Normierungs-DLL gehört prinzipiell zu einer Kanal-DLL, sie soll genau wie diese einfach ins Gesamtsystem hinzugefügt bzw. wieder entfernt werden können. Trotzdem hat sie keine direkte Schnittstelle zu der zugehörigen Kanal-DLL.

Diese Unterlage beschreibt die Einbindung und Schnittstelle jeder Normierungs-DLL zu den WinCC-Applikationen AlarmLogging und TagLogging. Sie ist entstanden bei dem Entwurf der S7PMC-Normierungs-DLL, deshalb ist der Begriff „S7PMC- Normierungs-DLL" größtenteils gleichzusetzen mit dem Begriff „Normierungs-DLL".

Prinzipieller Ablauf

Die S7PMC-Normierungs-DLL ist ein passives Programmgebilde, das ausschließlich zu den Applikationen AlarmLogging und TagLogging Schnittstellen besitzt. Die S7PMC-Normierungs-DLL bearbeitet S7PMC-spezifische Funktionen für AlarmLogging und TagLogging.

AlarmLogging und TagLogging melden sich mit einem Startaufruf bei der Normierungs-DLL an. Dabei werden bestimmte Parameter in einer Startstruktur an die Norm.DLL weitergegeben sowie deren Eigenschaften anhand von Kennungen entgegengenommen.

Zur Bearbeitung der S7PMC-Funktionen im Runtime-Modus sind zwei Datentransferrichtungen erforderlich:

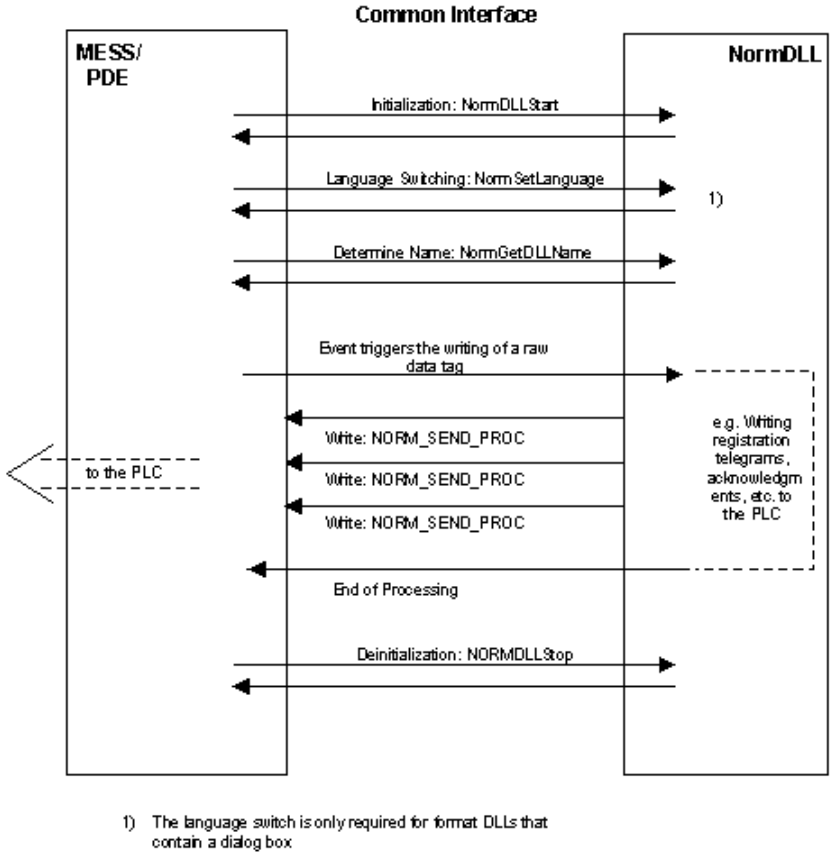
- OS zu AS: (Abgabe von An-/Abmeldeaufträgen, Quittierungen)
- AS zu OS: (Empfang von Meldungen und Archivdaten)

Mit einem Initialisierungsaufruf teilt TagLogging/AlarmLogging der S7PMC-DLL die projektierten Archivvariablennamen / Meldungsnummern mit. Für diese Objekte muß sich die Normierungs-DLL (WinCC) bei der AS anmelden. Der Initialisierungsaufruf kann zu jedem beliebigen Zeitpunkt bearbeitet werden,

Die Normierungs-DLL wird von AlarmLogging/TagLogging zur Deinitialisierung aufgerufen, um Resourcen zurückzugeben usw.

5.3.1 Gemeinsame Schnittstelle zu AlarmLogging und TagLogging

Die allgemeinen Funktionen der Normierungs-DLL, die für AlarmLogging und TagLogging identisch sind, sind in einer gemeinsamen Schnittstelle zusammengefaßt. Die Funktionsnamen beginnen alle mit 'NORM...'.
(Präfix für AlarmLogging-spezifische Funktionen: 'Mld...', Präfix für TagLogging-spezifische Funktionen: 'Pde...'.)

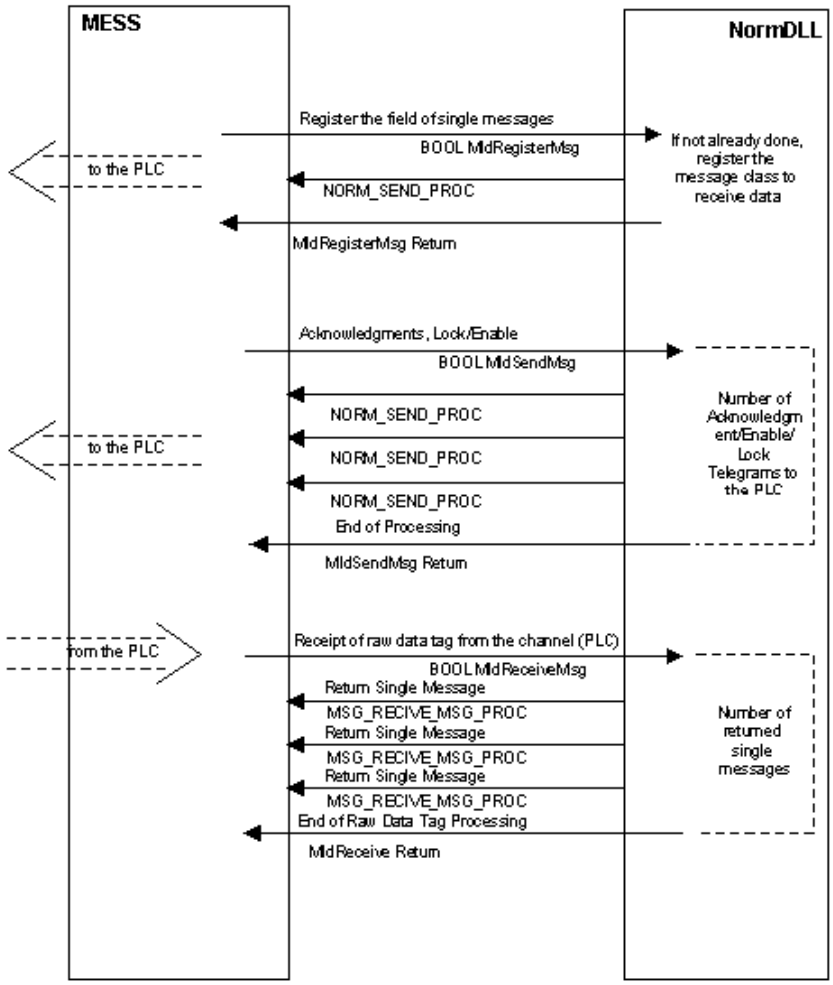


1) The language switch is only required for format DLLs that contain a dialog box

MELD = AlarmLogging
PDE = TagLogging

5.3.2 AlarmLogging-spezifische Zusätze

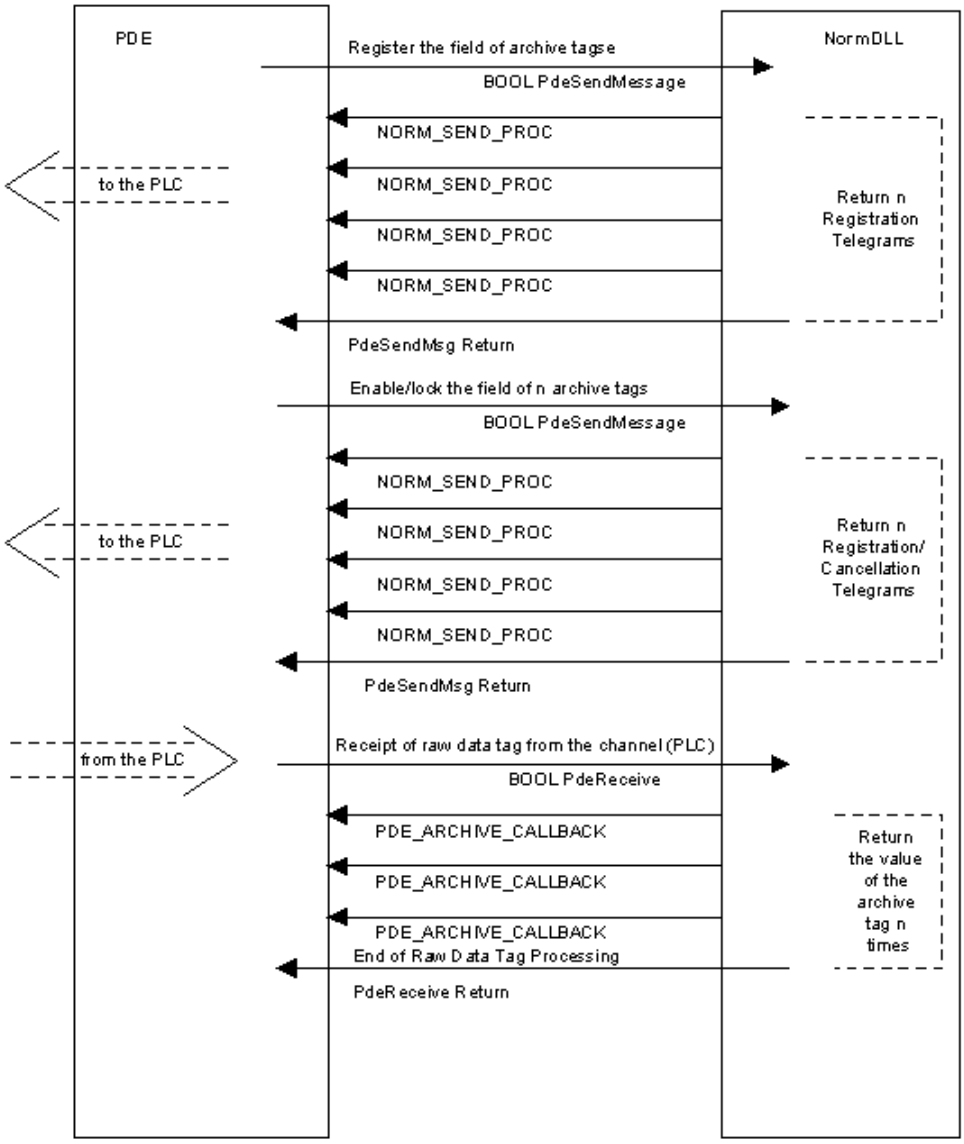
Runtime



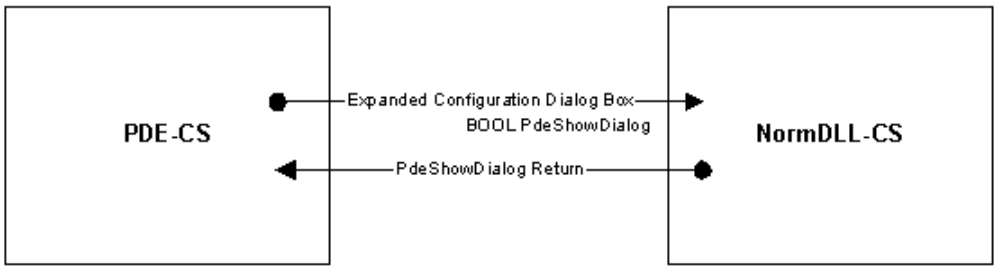
Erweiterter Projektierungsdialog

5.3.3 TagLogging-spezifische Zusätze

Runtime



Erweiterter Projektierungsdialog



5.3.4 API-Funktionen einer WinCC-Normierungs-DLL

Die Normierungs-DLL gliedert sich in folgende Teilbereiche:

- Initialisierung der Normierungs-DLL
  - Initialisierung durch das Betriebssystem während des Ladevorgangs der Normierungs-DLL (LibMain)
  - Abfrage der Eigenschaften einer Normierungs-DLL
  - Abfrage des Namens der Normierungs-DLL
- Shutdown der Normierungs-DLL
  - Shutdown durch TagLogging und AlarmLogging
  - Entladen durch das Betriebssystem
- Erweiterungen der Projektierung
  - Dialogerweiterung bei der Projektierung von Meldungen
  - Dialogerweiterung bei der Projektierung von Archivvariablen
- Online-Dienste
  - Registrierung aller Normierungs-DLL-spezifischen Objekte (Meldungen, Archivvariablen)
  - Sprachumschaltung
- Normierung
  - Normierung von Meldungen
  - Normierung von Archivvariablen

5.3.4.1 Initialisierung der Normierungs-DLL

Initialisierung während des Ladevorgangs

Die Applikationen AlarmLogging und/oder TagLogging laden eine WinCC-Normierungs-DLL mit Hilfe des Systemaufrufs *LoadLibrary*. Daraufhin wird die Normierungs-DLL durch das Betriebssystem geladen und durch dessen Standardmechanismen initialisiert. Alle Einsprungadressen der Normierungs-DLL sind festgelegt.

5.3.4.2 Abfrage der Eigenschaften einer Normierungs-DLL

Mit dem Aufruf NormDLLStart meldet sich AlarmLogging und TagLogging bei der jeweiligen Normierungs-DLL an. Er ist zum Informationsaustausch zwischen der Normierungs-DLL und der Applikation gedacht.

NormDLLStart

```
#include <winccnrm.h>

BOOL NormDLLStart(
        LPVOID lpUser,
        BOOL bModeRuntime,
        PNORM_STARTSTRUCT pcis,
        PCMN_ERROR lpError);
```

Parameter	Beschreibung
lpUser	Pointer auf Applikations-Daten, unverändert an Callback weitergeben
bModeRuntime	TRUE wenn Norm.-DLL im Runtime-Modus gestartet wird, FALSE wenn im Projektierungs-Modus; wird von Norm.-DLL derzeit nicht ausgewertet
pcis	Pointer auf Start-Struktur.
lpError	Pointer auf Standard-WinCC Fehlerstruktur

Return	Beschreibung
TRUE	Kein Fehler
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger lpError

NORM_STARTSTRUCT

Komponente	Beschreibung	E/A
dwSize	Größe der Struktur in Byte	A
lpstrProjectPath	Pfad des aktuell angewählten Projekts	E
NORM_SEND_PROC pfnWriteRwData	Ptr auf Callback-Funktion der Applikation, über die die Normierungs-DLL eine Rohdatenvariable über den DM an das AS schickt.	E
dwAppID	Applikationskennung: 1 = AlarmLogging 2 = TagLogging 3 = USER (für weitere Applikat. reserviert, z.Z. nicht benötigt)	E
dwLocaleID	zum Zeitpunkt des Aufrufs aktuelle Spracheinstellung	E
dwNormCap	Eigenschaften der Normierungs-DLL nach folgender Tabelle	A

Die Callback-Funktion zum Schicken von Rohdatenvariablen an den WinCC-Datenmananager wird wie folgt versorgt:

```
typedef BOOL(*NORM_SEND_PROC)(
        LPDM_VAR_UPDATE_STRUCT     lpDmVarUpdate,
        DWORD                      dwWait,
        LPVOID                     LpUser,
        LPCMN_ERROR                lpError );
```

Parameter	Beschreibung
lpDmVarUpdate	Zeiger auf Rohdatenvariable
dwWait	Kennung, ob die Applikation auf den Abschluß des Schreibaufrufs warten soll, oder nicht: WAIT_ID_NO mit SET_VALUE WAIT_ID_YES mit SET_VALUE_WAIT
lpUser	Pointer auf Applikations-Daten, beim Aufruf NormDLLStart gemerkt
lpError	Pointer auf Standard-WinCC Fehlerstruktur

Return	Beschreibung
TRUE	Kein Fehler
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger lpError

Jeder Eigenschaft ist ein Bit nach folgender Tabelle zugeordnet.

DEFINE	Bitmaske		Bedeutung
NORMCAP_DIALOG	0x00000001	gesetzt	Normierungs-DLL bietet spez. Dialog
		gelöscht	Normierungs-DLL bietet keinen Dialog
NORMCAP_REENTRANT	0x00000002	gesetzt	Normierungs-DLL ist renetrant-fähig
		gelöscht	Normierungs-DLL ist nicht renetrant-fähig
NORMCAP_MSG_FREE_LOCK	0x00000004	gesetzt	An-/Abmelden für Meldungen ist möglich.
		gelöscht	An-/Abmelden für Meldungen ist nicht möglich.
NORMCAP_ARC_FREE_LOCK	0x00000008	gesetzt	An-/Abmelden für Archivvariablen ist möglich.
		gelöscht	An-/Abmelden für Archivvariablen ist nicht möglich.
NORMCAP_MSG_GENERIC	0x00000010	gesetzt	Meldungen können generisch erzeugt werden.

DEFINE	Bitmaske		Bedeutung
		gelöscht	Meldungen können generisch nicht erzeugt werden.
NORMCAP_ARC_GENERIC	0x00000020	gesetzt	Archivvariablen können generisch erzeugt werden.
		gelöscht	Archivvariablen können generisch nicht erzeugt werden.

5.3.4.3 Abfrage des Namens der Normierungs-DLL

NormGetDLLName

```
#include <winccnrm.h>

LPTSTR NormGetDLLName( void );
```

Return	Beschreibung
LPTSTR	Zeiger auf einen String, der den Namen der Normierungs-DLL in Klartext enthält; der Name ist abhängig von der aktuellen Spracheinstellung.

5.3.5 Shutdown der Normierungs-DLL

Shutdown durch TagLogging und AlarmLogging

TagLogging und AlarmLogging verständigen die Normierungs-DLL, wenn die Applikationen beendet werden. In der Normierungs-DLL werden dann die Resourcen ordnungsgemäß zurückgegeben.

NormDLLStop

```
#include <winccnrm.h>

BOOL NormDLLStop (void);
```

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion

Entladen durch das Betriebssystem

Es sind keine besonderen Vorkehrungen erforderlich.

5.3.5.1 Erweiterungen der Projektierung

Für die S7PMC Objekte sind spezifische Angaben erforderlich. Diese Angaben werden zunächst in einem Dialog mit Standardmitteln (ohne MFC) erfragt und gehen direkt entweder in die WinCC-Meldungsnummer oder in den Namen der Archivvariablen ein. Damit muß die Normierungs-DLL diese Angaben nicht selbst ablegen und verwalten. Um die projektweite Eindeutigkeit einer Meldungsnummer oder Archivvariablen zu gewährleisten, ist eine Zuordnung zwischen Meldungsnummer bzw. Archivvariablen und der zugehörigen Rohdatenvariablen erforderlich. Diese Zuordnungsinformation ist Bestandteil der Meldungsnummer bzw. des Archivvariablennamens.

5.3.5.2 Dialogerweiterung bei der Projektierung von S7PMC-Meldungen

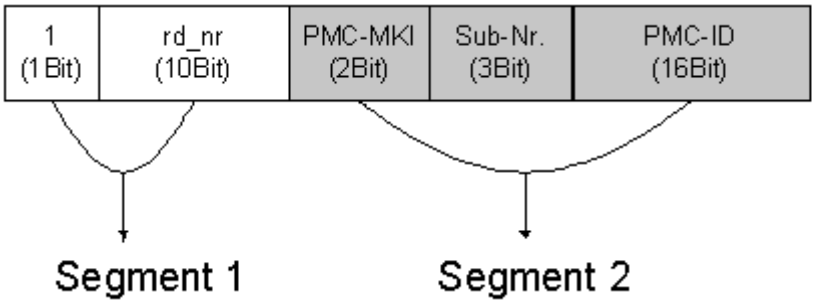
Die Normierungs-DLL hat eine API - Funktion zum Festlegen der S7PMC-spezifischen Meldungsnummer. Diese Funktion wird vom Meldesystem CS beim Parametrieren von Einzelmeldungen, die zu einer S7PMC-Normierungs-DLL gehören, aufgerufen. Als Meldungsnummer vergibt die S7PMC-Normierungs-DLL die Nummer, die aus zwei Teilen besteht:

Teil 1:
Nummer die projektweit eine AG-CPU eindeutig identifiziert (Rohdatenvariablen-Nummer)

Teil 2:
Nummer die AS-seitig zu der Meldung gehört und diese eindeutig innerhalb einer AG-CPU identifiziert (Normierungs-DLL-spezifisch).

Im Projektierungsdialog muß folgende Auswahl getroffen werden, um die Meldungsnummer aufzubauen:

Struktur eines S7PMC-Meldungsnummer (32 Bit)



Zu Teil 1

Jede Meldung gehört zu einer Rohdatenvariablen, die eine AG-CPU identifiziert. Um die Zuordnung Rohdatenvariable – Meldungsnummer vornehmen zu können, wurde folgende Festlegung getroffen.
Der Name der Rohdatenvariable für S7PMC - und allen Verbindungsarten mit Normierungs-DLL - hat folgenden festen Aufbau:

@rd_alarm#rd_nr

@rd_alarm#	fester Bestandteil des Namens einer Rohdatenvariablen für Normierungs-DLL´s
rd_nr	Dezimalzahl von 0 - 1023 zur Identifikation einer Rohdatenvariablen (ohne führende Nullen)

Das höchstwertigste Bit der Meldungsnummer ist bei Meldungsnummern die von Normierungs-DLL's (extern) vergeben werden gesetzt. Diese Meldungen dürfen nur von den zugehörigen Normierungs-DLL`s bearbeitet werden, d. h. über den Projektierungsdialog von AlarmLogging kann die Meldungsnummer nicht verändert werden.

Zu Teil 2

Dieser Teil der Meldungsnummer kann von der jeweiligen Normierungs-DLL belegt werden. Bei S7PMC hat er folgende Bedeutung:

MKl	Meldungsklasse; auszuwählen ist eine der Klassen: SCAN (1) ALARM/NOTIFY (2) ALARM_8P/ALARM_8 (2) LTM (3)
Sub-Nr	Submeldungsnummer nur relevant bei ALARM_8 und ALARM_8P: 1...8
PMC-ID	PMC-Meldungsnummer (Baustein-Eingangsparameter EV-ID): 1...16386 bei Meldungsklasse SCAN und ALARM/NOTIFY bzw. ALARM_8P/ALARM_8 1...7 bei Meldungsklasse LTM

MldShowDialog

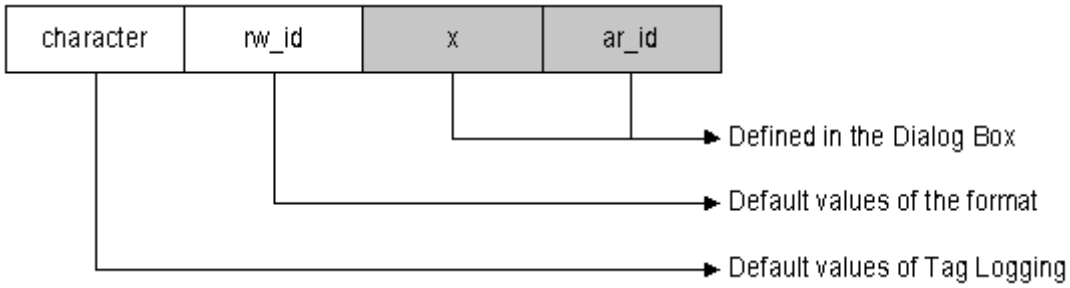
```
#include <winccnrm.h>

BOOL WINAPI MldShowDialog(
              HWND                  hwnd,
              LPMSG_CSDATA_GENERIC  lpmCS,
              LPDM_PROJECT_INFO     lpDMProjectInfo,
              LPCMN_ERROR           lpError );
```

Parameter	Beschreibung
hwnd	Fensterhandle
lpmCS	Zeiger auf Einzelmeldungsdaten
lpDMProjectInfo	Zeiger auf ProjektInfostruktur
lpError	Zeiger auf Standard-WinCC Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

5.3.5.3 Dialogerweiterung bei der Projektierung von Archivvariablen

Die Normierungs-DLL hat eine API - Funktion zum Festlegen der S7PMC-spezifischen Archivvariablennamen. Diese Funktion wird vom TagLogging CS beim Parametrieren von Archivvariablen, die zu einer S7PMC-Verbindung gehören, aufgerufen. Als Archivvariablenname vergibt die S7PMC-Normierungs-DLL einen Namen, der aus mehreren Komponenten besteht, u.a. enthält er die AS-seitig zu dem Archiv gehörende Nummer. Mit diesem Algorithmus sind die S7PMC-Archivnummern eindeutig in der WinCC-Archivvariablenbeschreibung enthalten, was im Runtimebetrieb zu einer schnellstmöglichen Zuordnung führt.
TagLogging garantiert, daß die Archivvariablennamen insgesamt eindeutig sind.
Struktur eines S7PMC-Archivvariablennamens (max. 18 Byte lang)



Bezeich-nung	Länge in Byte	Vergeben von	Bedeutung
zeichen	9	TagLogging	Feste Zeichenfolge, die von TagLogging vergeben wird, besteht aus dem Namen der Normierungs-DLL und # als Trennungszeichen. z. B. für S7PMC: NRMS7PMC, erscheint nicht an der Oberfläche
rw_id	8	TagLogging/ Norm.DLL	Rohdaten-ID in Hexa-Zeichen (inklusive führende Nullen), damit findet eine eindeutig Zuordnung zu der Rohdatenvariablen (Verbindung) statt, zu der die Archivnummer gehört. Der Namensanteil wird von der Norm.DLL anhand der TagLogging-Eingangsparameter gebildet.
x	1	Norm.DLL-CS-Anteil	S7PMC-spezifische Kennung zur Unterscheidung zwischen BSEND und AR_SEND: 'A' = AR_SEND 'B' = BSEND
ar_id	4	Norm.DLL-CS-Anteil	ID als Hexa-Zeichen (inklusive führende Nullen) Abhängig von Kennung x: S7PMC-spezifische Archivnummer AR_ID oder S7-spezifische R_ID beim BSEND

Beispiel eines für S7PMC erzeugten Archivvariablennamens: #00000001#A#0014

PdeShowDialog

```
#include <winccnrm.h>

BOOL WINAPI PdeShowDialog(
                LPVOID          hwnd,
                LPTSTR          lpszArcVarName,
                DWORD           dwArcVarNameLength,
                LPDM_VARKEY     lpVarKey,
                LPCMN_ERROR     lpError
                );
```

Parameter	Beschreibung
hwnd	Fensterhandle
lpszArcVarName	Zeiger auf Stringfeld zur Ablage des Normierungs-DLL-spez. Archivvariablen-Namenanteils
dwArcVarNameLength	max. Länge Normierungs-DLL-spez. Namenanteils
lpVarKey	Zeiger auf Varkey der Rohdatenvariablen
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

5.3.5.4 Online-Dienste

Alle Meldungen registrieren

Diese Funktion ist erforderlich, weil die Normierungs-DLL keine Projektierungsinformation über die relevanten Meldungen hat. Meldungen werden von dem AS aber erst geschickt, wenn sich die Applikation (WinCC) zum Empfang von Meldungen angemeldet hat. AlarmLogging ruft die Funktion MldRegisterMsg derzeit für jede relevante Meldung auf und übergibt damit der Normierungs-DLL Projektierungsinformation für die Einzelmeldung. Außer der Meldungsbeschreibung erhält die Normierungs-DLL einen Zeiger auf die Rohdatenvariable (Verbindung) der dieser Meldung zugeordnet ist. Damit kann die Normierungs-DLL zur Laufzeit im Hauptspeicher eine Tabelle anlegen, mit der die S7PMC-spezifischen Anmeldetelegramme aufgebaut werden können.

MldRegisterMsg

```
#include <winccnrm.h>

BOOL WINAPI MldRegisterMsg(
              LPDM_VARKEY   lpDMVarKey,
              LPDWORD       lpMsgNumber,
              DWORD         DwNumMsgNumber,
              LPCMN_ERROR   lpError );
```

Parameter	Beschreibung
lpDMVarKey	Zeiger auf Varkey der Rohdatenvariablen
lpMsgNumber	Zeiger auf Feld mit Einzelmeldungsnummern
dwNumMsgNumber	Anzahl Einzelmeldungsnummern
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

5.3.5.5 Registrieren aller Archivvariablen

Diese Funktion ist erforderlich, weil die Normierungs-DLL keine Projektierungsinformation über die relevanten Archivvariablen hat. Deshalb wird die Funktion PdeSendMsg für eine bestimmte Anzahl relevanter Archivvariablen aufgerufen und damit die Projektierungsinformation und TagLogging-Zusatzinformation für die Archivvariablen bekanntgegeben.
Es können mehrere Archivvariablen einer Verbindung mit einem Aufruf registriert werden.

Pro Archivvariablen übergibt TagLogging ein Doppelwort als Zusatzinformation an die Normierungs-DLL, die im Speicher der Normierungs-DLL gehalten wird. Diese Zusatzinformation wird von TagLogging benötigt, sobald Archivvariablen-Werte zu bearbeiten sind (in der Callback-Funktion TagLogging_ARCHIVE_CALLBACK). Damit kann die Normierungs-DLL zur Laufzeit im Hauptspeicher eine Tabelle anlegen, mit der die S7PMC-spezifischen Anmeldetelegramme für die jeweiligen Archive aufgebaut werden können. Die Anmeldetelegramme sind erforderlich, um dem AS die Empfangsbereitschaft für die jeweilige Archivnummer bekanntzugeben. Erst nach erfolgreichem Anmelden schickt das AS die Archivdaten an die Applikation (WinCC).

PdeSendMsg

```
#include <winccnrm.h>

BOOL WINAPI PdeSendMsg(
      NORM_SEND_PROC     lpfnCallBack,
      DWORD              dwFunctionId,
      LPSZ_ARC_VAR_NAME  lpszArcVarName,
      LPDWORD            lpdwData,
      DWORD              dwNumArchVarName,
      LPDM_VARKEY        dmVarKey,
      LPVOID             lpUser,
      LPCMN_ERROR        lpError
      );
```

Parameter	Beschreibung
lpfnCallBack	Zeiger auf Callback-Routine mit der die von der Normierungs-DLL aufgebaute Rohdatenvariable an den DM zu übergeben ist. Falls Null, wird die Callback-Routine aus der Ini-Struktur aufgerufen. Die Funktions-Adresse aus der Ini-Struktur ist nicht identisch mit diesem Parameter.
dwFunctionId	Funktionskennung FUNC_ID_REGISTER (vgl. folgende Tabelle), dieselbe Funktion gilt für alle aufgeführten Variablen
lpszArcVarName	Zeiger auf ein Pointerfeld, deren Elemente auf die Namen der Archivvariablen verweisen
lpdwData	Zeiger auf ein Feld dessen Elemente Zusatzdaten für die Archivvariablen enthalten, kann auch Null sein. Der einer Archivvariablen zugehörige Zusatzwert wird bei der Funktion FUNC_ID_REGISTER (Archivvariable anmelden) unverändert in interne Listen der Normierungs-DLL übernommen und zu gegebener Zeit an die TagLogging_ARCHIVE_CALLBACK weitergereicht. Bei den anderen Funktionskennungen ohne Bedeutung.
dwNumArchVarName	Anzahl Archivvariablennamen, die zu bearbeiten sind
lpVarKey	Zeiger auf Varkey der Rohdatenvariablen
lpUser	Zeiger auf Userdaten, unverändert an Callback übergeben
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

Mögliche Funktionen der Prozedur PdeSendMsg (Werte von dwFunctionId):

DEFINE	Bitmaske	Bedeutung
FUNC_ID_ LOCK	0x00000001	Archivvariable sperren
FUNC_ID_ FREE	0x00000002	Archivvariable freigeben
FUNC_ID_ REGISTER	0x00000004	Archivvariable anmelden
FUNC_ID_UN REGISTER	0x00000008	Archivvariable abmelden (wird derzeit nicht benötigt)

5.3.5.6 Sprachumschaltung

Der Projektierungsdialog muß sprachabhängig sein, d. h. die Normierungs-DLL muß die aktuell eingestellte Sprache kennen. Beim Start wird die Spracheinstellung in der Startstruktur mitgegeben. Die dynamische Sprachumschaltung muß von TagLogging und AlarmLogging auch an die Normierungs-DLL weitergereicht werden. Dafür gibt es den Aufruf

NormSetLanguage

```
#include <winccnrm.h>

BOOL NormSetLanguage(
                DWORD           dwLocaleID,
                LPCMN_ERROR     lpError
                );
```

Parameter	Beschreibung
dwLocaleID	zum Zeitpunkt des Aufrufs aktuelle Spracheinstellung
lpError	Zeiger auf Standard-WinCC Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

5.3.6 Normierung

Hat sich eine Applikation für den Empfang von Meldungen oder Archivdaten beim AS angemeldet, erhält sie diese Daten über die jeweilige Rohdatenvariable. Das Anmelden führt die Normierungs-DLL beim Registrieren durch. Ab diesem Zeitpunkt können Datentelegramme vom AS kommen. Die Datentelegramme werden in Rohdatenvariablen verpackt und über die Kanal-DLL, den Datenmanager und die jeweilige Applikation (hier TagLogging oder AlarmLogging) an die Normierungs-DLL weitergereicht, die für den Typ der Rohdatenvariablen zuständig ist. Die Normierungs-DLL interpretiert die ankommenden Daten und bildet daraus Meldungen bzw. Archivdaten.

5.3.6.1 Ableitung von Einzelmeldungen

In dem Inhalt einer Rohdatenvariablen (eines Telegramms) können *n* Einzelmeldungen abgelegt sein. Die Normierungs-DLL muß dieses S7PMC-spezifisches Telegramm interpretieren und die sich daraus ergebenden Einzelmeldungen an AlarmLogging weiterreichen.

Die Meldungsnummer (EV_ID) von S7PMC ist ein Teil der WinCC-Meldungsnummer. Mit einer Meldung können von S7PMC bis zu max. 10 Prozeßwerte mitgeliefert werden. Dabei ist als Prozeßwert auch der Typ „string“ erlaubt. Diesen Prozeßwerttyp unterstützt AlarmLogging nicht, derartige Zusatzwerte muß die Normierungs-DLL verwerfen.

Die Funktion MldReceiveMsg wird von AlarmLogging auch jedes Mal aufgerufen, wenn sich der Status der Rohdatenvariablen geändert hat, also wenn Status gestört nach OK oder umgekehrt vom DM festgestellt wird. Die Status-(Zustands-)Änderung der Rohdatenvariablen (entspricht einer Verbindung) ist für die S7PMC-Normierungs-DLL von Bedeutung. Näheres dazu ist im Kapitel „Bearbeitung beim Zustandswechsel“ beschrieben.

MldReceiveMsg

```
#include <winccnrm.h>

BOOL WINAPI MldReceiveMsg(
              MSG_RECEIVE_MSG_PROC     lpfnMsgReceive,
              LPDM_VAR_UPDATE_STRUCT   lpDMVar,
              LPVOID                   lpUser,
              LPCMN_ERROR              lpError );
```

Parameter	Beschreibung
lpfnMsgReceive	Zeiger auf Callback-Routine mit der die von der Normierungs-DLL aufgebaute Einzelmeldung an AlarmLogging zu übergeben ist.
lpDMVar	Zeiger auf Rohdatenvariable
lpUser	Zeiger auf Userdaten, unverändert an Callback übergeben
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

Die Callback-Funktion zum Abgeben der Einzelmeldungen an AlarmLogging wird wie folgt versorgt:

```
typedef BOOL (*MSG_RECEIVE_MSG_PROC)(
                    LPMSG_RTCREATE_STRUCT   lpMsgCreate,
                    DWORD                   dwNumMsg,
                    LPVOID                  lpUser,
                    LPCMN_ERROR             lpError);
```

Parameter	Beschreibung
lpMsgCreate	Zeiger auf eine WinCC - Meldung
dwNumMsg	Anzahl Einzelmeldungen
lpUser	Pointer auf Applikations-Daten
lpError	Pointer auf Standard-WinCC Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

5.3.6.2 Meldungen quittieren, sperren/freigeben

Das Melde- und Quittierkonzept von WinCC-AlarmLogging und S7PMC sieht vor, daß Meldungen, abhängig von ihrer Projektierung, quittiert werden können. Die Quittungsinformation ist AlarmLogging bekannt, muß aber auch im Meldequittierspeicher des AS geführt werden. Um das zu erreichen, schickt AlarmLogging Quittungstelegramme über die der Verbindung entsprechenden Normierungs-DLL an das AS.
Die S7PMC-Normierungs-DLL baut anhand dieser Eingangsdaten die zugehörigen S7PMC-Telegramme auf, die über die AlarmLogging-Callback-Funktion NORM_SEND_PROC an den DM weitergereicht werden.
Dasselbe Verfahren gilt wenn eine Einzelmeldung von AlarmLogging gesperrt / wieder freigegeben - d. h. ihre Erzeugung an der Quelle im AS wird unterbunden / wieder freigeschaltet - werden soll.

MldSendMsg

```
#include <winccnrm.h>

BOOL WINAPI MldSendMsg(
                NORM_SEND_PROC          lpfnMsgSend,
                LPMSG_SEND_DATA_STRUCT  lpSendData,
                DWORD                   dwNumData,
                LPVOID                  lpUser,
                LPCMN_ERROR             lpError );
```

Parameter	Beschreibung
lpfnMsgSend	Zeiger auf AlarmLogging-Callback-Routine, mit der die von der Normierungs-DLL aufgebaute Rohdatenvariable zum Schreiben an das AS zu übergeben ist. Die Parameter sind in Kap. „Abfrage der Eigenschaften einer Normierungs-DLL“ beschrieben

Parameter	Beschreibung
lpSendData	Zeiger auf Sendedaten, Aufbau nachfolgend beschrieben
dwNumData	Anzahl zu bearbeitender Einzelaufträge
lpUser	Zeiger auf Userdaten, unverändert an Callback übergeben
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

Aufbau der AlarmLogging-Sendedaten (Einzelauftrag)

Variable	Bedeutung
DWORD dwVarID	Rohdatenvariablen - ID des DM
DWORD dwNotify	Notify Mögliche Werte MSG_STATE_QUIT Meldung quittieren MSG_STATE_LOCK Meldung sperren MSG_STATE_UNLOCK Meldung freigeben MSG_STATE_QUIT_EMERGENCY Alle Meldungen quittieren
DWORD dwData	bei QUIT, LOCK, UNLOCK --> Meldungsnummer bei NOTQUIT --> unbenutzt

5.3.6.3 Bearbeitung beim Zustandswechsel

Der Statuswechsel einer Verbindung (Rohdatenvariablen) muß der Normierungs-DLL bekannt gegeben werden. Dies geschieht über die Funktion **MldReceiveMsg**.

Statuswechsel von - nach	Bearbeitung in der S7PMC-Normierungs-DLL
gestört - OK	Anmeldetelegramme für alle S7PMC-Meldungsklassen, für die mindestens eine Meldung projektiert wurde, an das AS abgeben. Die Anmeldung wird S7PMC-Meldungsklassen-spezifisch vorgenommen. Die Normierungs-DLL kennt schon alle projektierten Meldungen aufgrund derer Registrierung.
OK - gestört	Die Normierungs-DLL muß aktive Aufträge verwerfen, die schon an das AS geschickt wurden, aber wegen des Zustandswechsels nicht mehr vollständig bearbeitet werden können (Quittungen fehlen).

5.3.6.4 Meldungsupdate der S7PMC-Normierungs-DLL

Beim Meldungsupdate liest die S7PMC-Normierungs-DLL den Meldezustand aller ihr durch das Registrieren bekanntgegebenen Meldungen und schickt ihn als Einzelmeldung an AlarmLogging. Somit kann beim Systemanlauf auf ein konsistentes Meldebild aufgesetzt werden.

Ein Meldungsupdate ist erforderlich, wenn

- ein Zustandswechsel von gestört nach OK erkannt wurde (das ist implizit auch beim Systemanlauf der Fall)
- das AS ein „Meldungsupdate-Telegramm“ schickt. Dieses Telegramm wird an jeden angemeldeten Teilnehmer geschickt, wenn z. B. ein Meldungsüberlauf festgestellt wird, wenn Meldungen von anderen Teilnehmern quittiert oder freigegeben werden.

Beim Meldungsupdate schickt das AS die Melde-Quittierzustände und die Sperrkennungen. Die Zusatzwerte und die Uhrzeit der Meldungen werden nicht geschickt. Die Normierungs-DLL versorgt in diesem Fall die Uhrzeit der Einzelmeldung mit der aktuellen Systemzeit und odert in den Meldestatus die Kennung MSG_STATE_UPDATE.

5.3.6.5 Normierung von Archivvariablen

Für TagLogging stellt die Normierungs-DLL zwei Funktionen zur Verfügung:

- Ableitung von einzelnen Archivvariablen-Werten aus dem Inhalt einer Rohdatenvariablen
- Sperren / Freigeben von Archivvariablen

5.3.6.6 Ableitung von einzelnen Archivvariablen-Werten

In dem Inhalt einer Rohdatenvariablen (eines Telegramms) können n Archivvariablen-Werte abgelegt sein. Die Normierungs-DLL muß dieses S7PMC-spezifisches Telegramm interpretieren und die sich daraus ergebenden Archivvariablen-Werte an TagLogging weiterreichen.

Für eine Archivvariable können Prozeßwertumformer mitgeschickt werden. Die S7PMC-Normierungs-DLL nimmt dann die gewünschte Umrechnung von Prozeßwert in Archivvariablen-Wert vor. Dabei werden in WinCC schon existierende Skalierungsfunktionen eingesetzt. Die genaue Vorgehensweise muß noch festgelegt werden.

Die Funktion PdeReceive wird von TagLogging auch jedes Mal aufgerufen, wenn sich der Status der Rohdatenvariablen geändert hat, also wenn Status gestört nach OK oder umgekehrt vom DM festgestellt wird. Die Status-(Zustands-)Änderung der Rohdatenvariablen (entspricht einer Verbindung) ist für die S7PMC-Normierungs-DLL von Bedeutung. Näheres dazu ist im Kapitel „Bearbeitung beim Zustandswechsel“ beschrieben.

PdeReceive

```
#include <winccnrm.h>

BOOL PdeReceive (
                LPDM_VAR_UPDATE_STRUCT          lpDmVarUpdate,
                TagLogging_ARCHIVE_CALLBACK     lpfnCallBack,
                LPVOID                          lpUser,
                LPCMN_ERROR                     lpError
                );
```

Parameter	Beschreibung
lpDmVarUpdate	Zeiger auf Rohdatenvariable
lpfnCallBack	Zeiger auf Callback-Routine, mit der die Normierungs-DLL die einzelnen Archivvariablen-Werte an TagLogging übergibt.
LpUser	Zeiger auf Userdaten, unverändert an Callback übergeben
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

Die Callback-Funktion zum Abgeben der einzelnen Archivvariablen-Werte an TagLogging wird wie folgt versorgt:

```
BOOL (*PDE_ARCHIVE_CALLBACK) (
                LPTSTR          lpszArcVarName,
                double          doValue,
                SYSTEMTIME*     lpstTime
                DWORD           dwFlags
                DWORD           dwData,
                LPVOID          lpUser,
                LPCMN_ERROR     lpError
                );
```

Parameter	Beschreibung
lpszArcVarName	Archivvariablen-Name ab Rohdaten-ID
doValue	Archivvariablen-Wert
lpstTime	Zeiger auf Zeitstempel, aus den Nutzdaten der Rohdatenvariablen abgeleitet
dwFlags	Kennungen, deren genaue Bedeutung noch festgelegt wird.
dwData	Zusatzdatum, das beim Registrieren der Archivvariablen mitgegeben wurde, unverändert übergeben
lpUser	Zeiger auf Userdaten, unverändert von Funktionsaufruf übernommen
lpError	Zeiger auf WinCC-Fehlerstruktur

Return	Beschreibung
TRUE	Funktion erfolgreich
FALSE	Fehler in API-Funktion, Beschreibung der Fehlerursache über den Zeiger `lpError`

5.3.6.7 Sperren / Freigeben von Archivvariablen

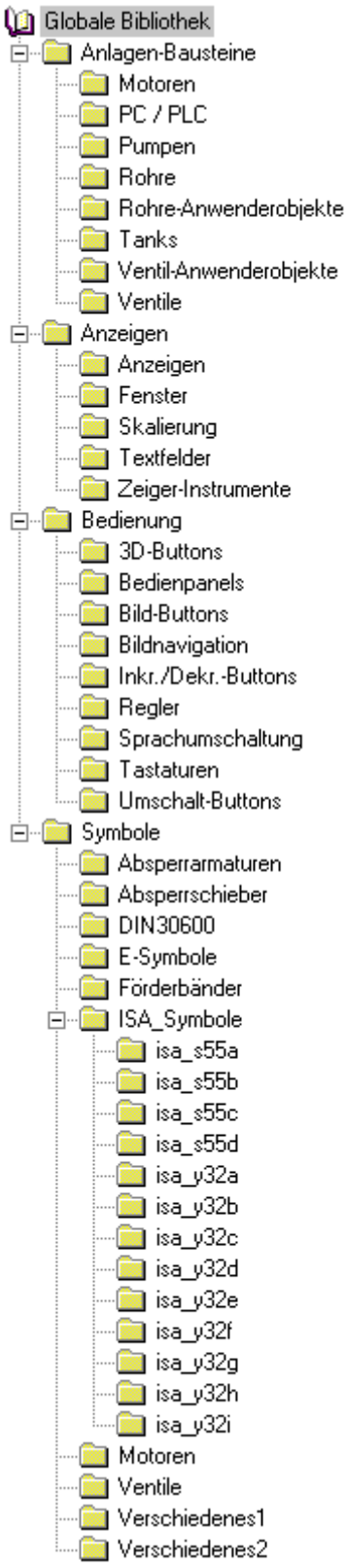
Mit dieser Funktion nutzt TagLogging die Möglichkeit in S7PMC, den Empfang von Archivvariablenwerten zu steuern. Die S7PMC-Normierungs-DLL bildet dann einen Aufruf zum Abmelden bzw. wieder Anmelden des jeweiligen Archivs und gibt diesen Aufruf über die NORM_SEND_PROC an den DM weiter.
Das Sperren / Freigeben von Archivvariablen ist aus der Sicht der S7PMC-Normierungs-DLL fast identisch mit den Funktionen, die beim Registrieren einer Archivvariablen erforderlich sind. Für beide Funktionen wird deshalb dieselbe Funktion in der Normierungs-DLL aufgerufen PdeSendMsg.
Über die Funktionskennung dwFunctionId wird zwischen Registrieren und Sperren / Freigeben unterschieden: beim Sperren / Freigeben haben die Zusatzdaten pro Archivvariable lpdwData keine Bedeutung. Vgl. Kap. „Registrieren aller Archivvariablen“.

5.3.6.8 Bearbeitung beim Zustandswechsel

Der Statuswechsel einer Verbindung (Rohdatenvariablen) muß der Normierungs-DLL bekannt gegeben werden, dies geschieht über die Funktion **PdeReceive**.

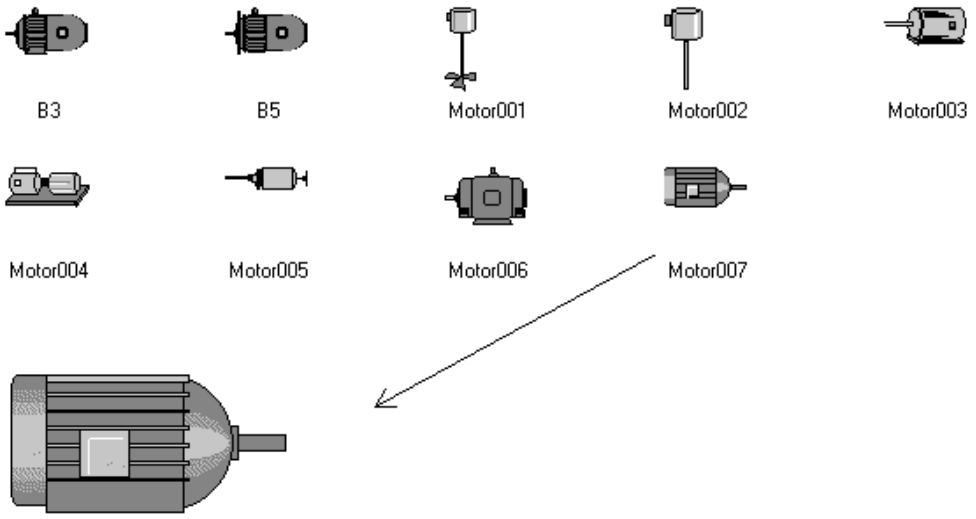
Statuswechsel von - nach	Bearbeitung in der S7PMC-Normierungs-DLL
gestört - OK	Anmeldetelegramme für alle Archivvariablen aller Verbindungen Die Normierungs-DLL kennt schon alle projektierten Archivvariablen aufgrund derer Registrierung.
OK - gestört	Die Normierungs-DLL muß aktive Aufträge verwerfen, die schon an das AS geschickt wurden aber wegen des Zustandswechsels nicht mehr vollständig bearbeitet werden können (Quittungen fehlen).

5.4 Globale Bibliothek

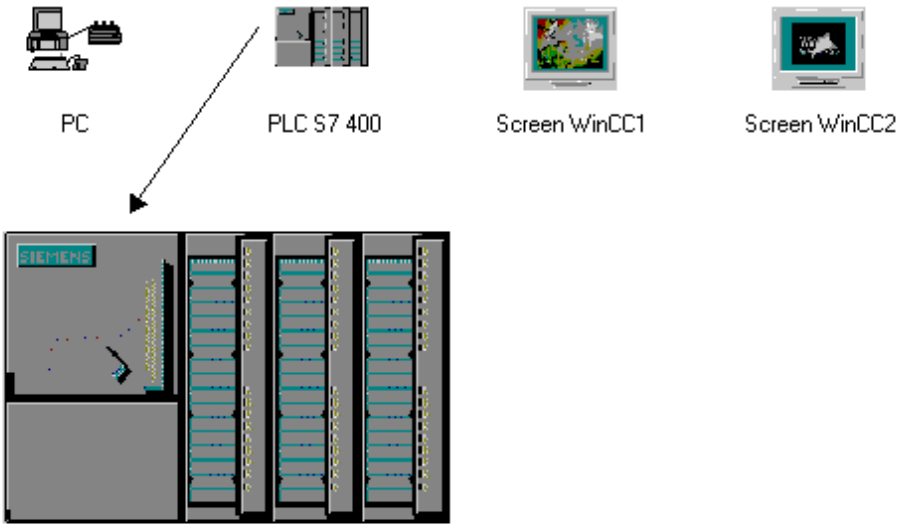


5.4.1 Anlagenbausteine

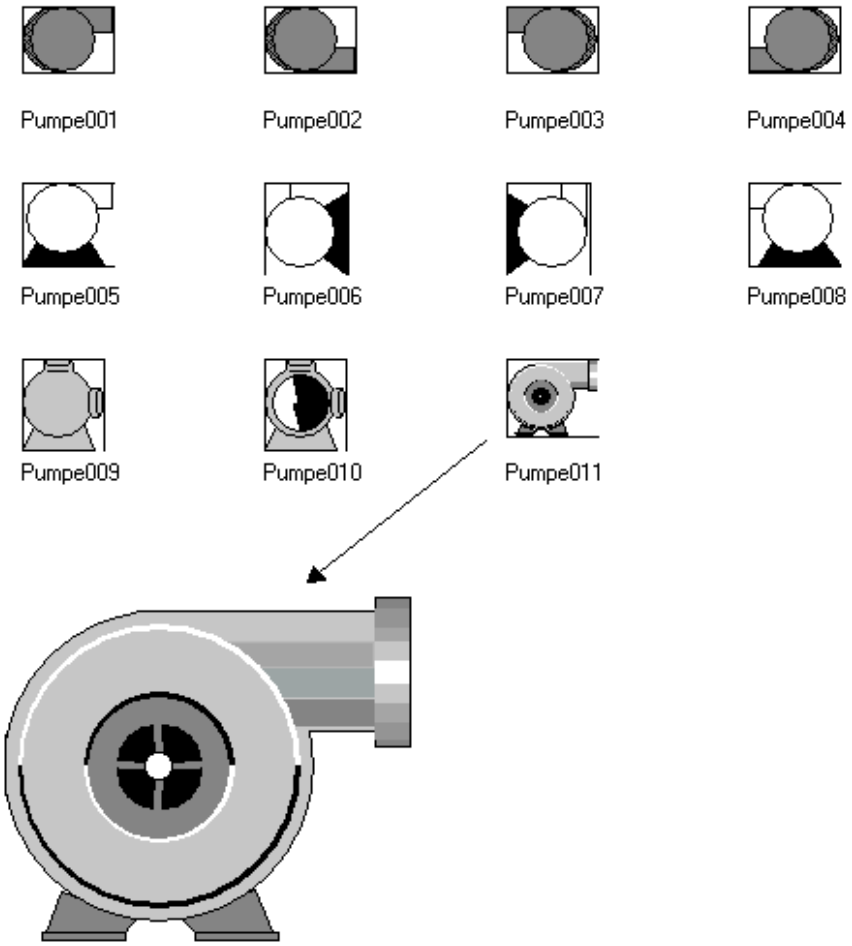
5.4.1.1 Motoren



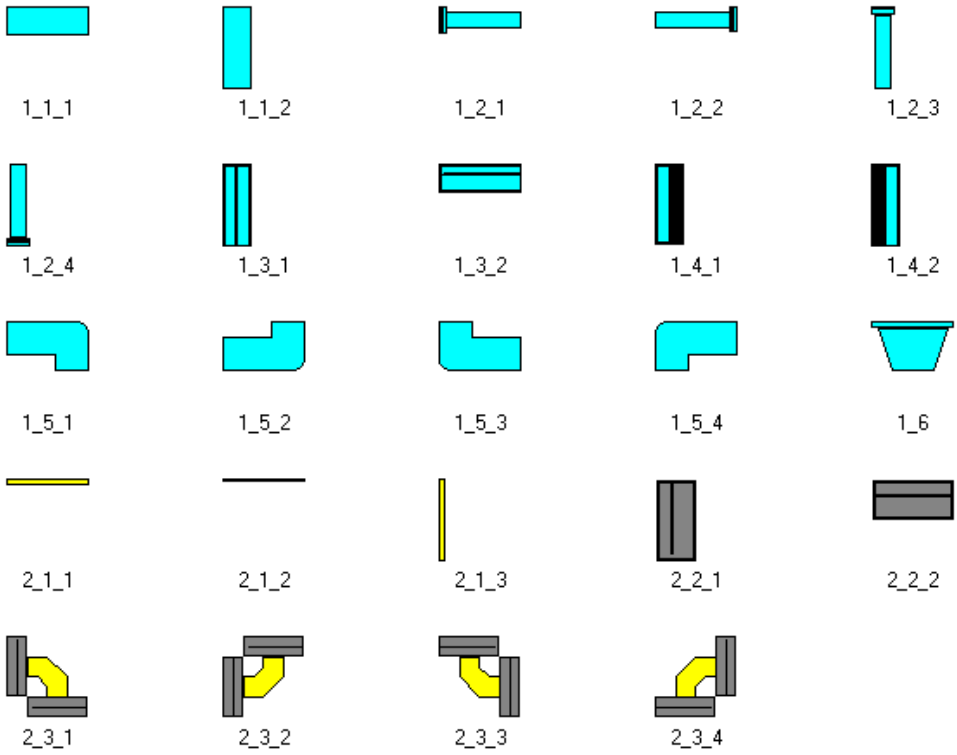
5.4.1.2 PC / PLC



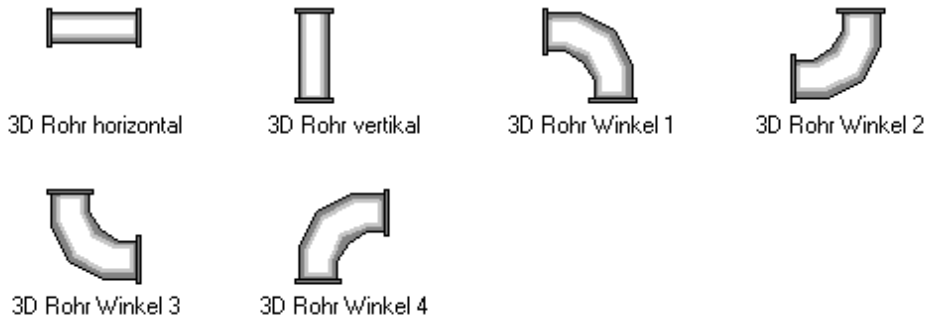
5.4.1.3 Pumpen


Pumpe001
Pumpe002
Pumpe003
Pumpe004
Pumpe005
Pumpe006
Pumpe007
Pumpe008
Pumpe009
Pumpe010
Pumpe011

5.4.1.4 Rohre



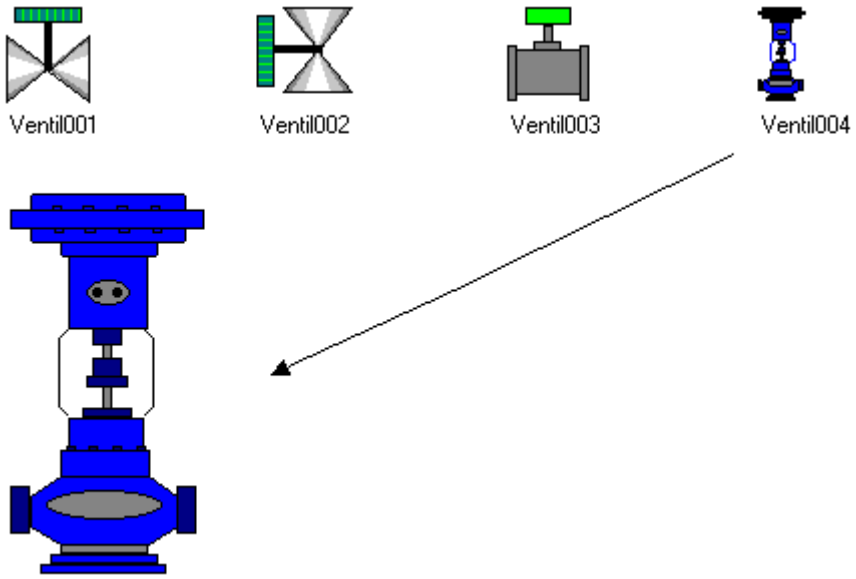
5.4.1.5 Rohre – Anwenderobjekte



5.4.1.6 Tanks

5.4.1.7 Ventil Anwender Objekte

5.4.1.8 Ventile



5.4.2 Anzeigen

5.4.2.1 Anzeigen

8-Bit Anzeige

8-Bit Anzeige + I/O Field

Digitalausgabe

5.4.2.2 Fenster

1

2

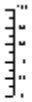
3

4

5

6

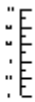
5.4.2.3 Skalierung



01

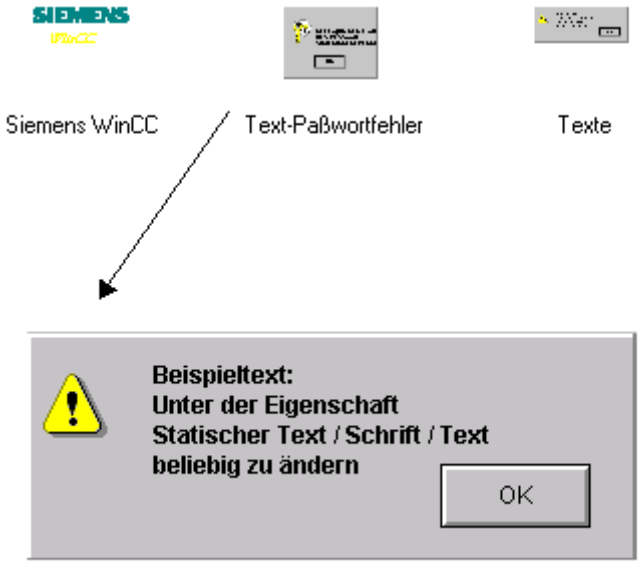
02

03



04

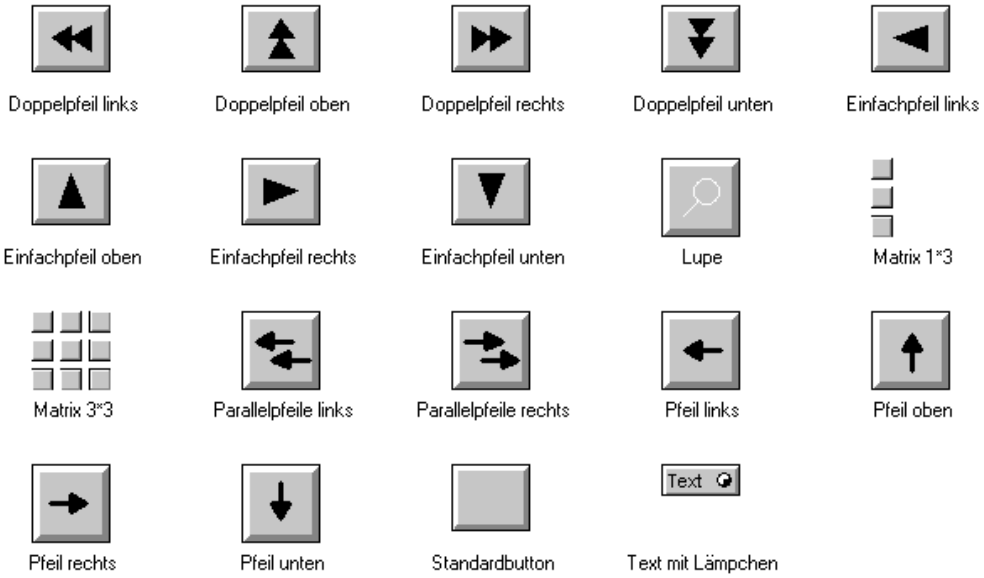
5.4.2.4 Textfelder



5.4.2.5 Zeiger-Instrumente

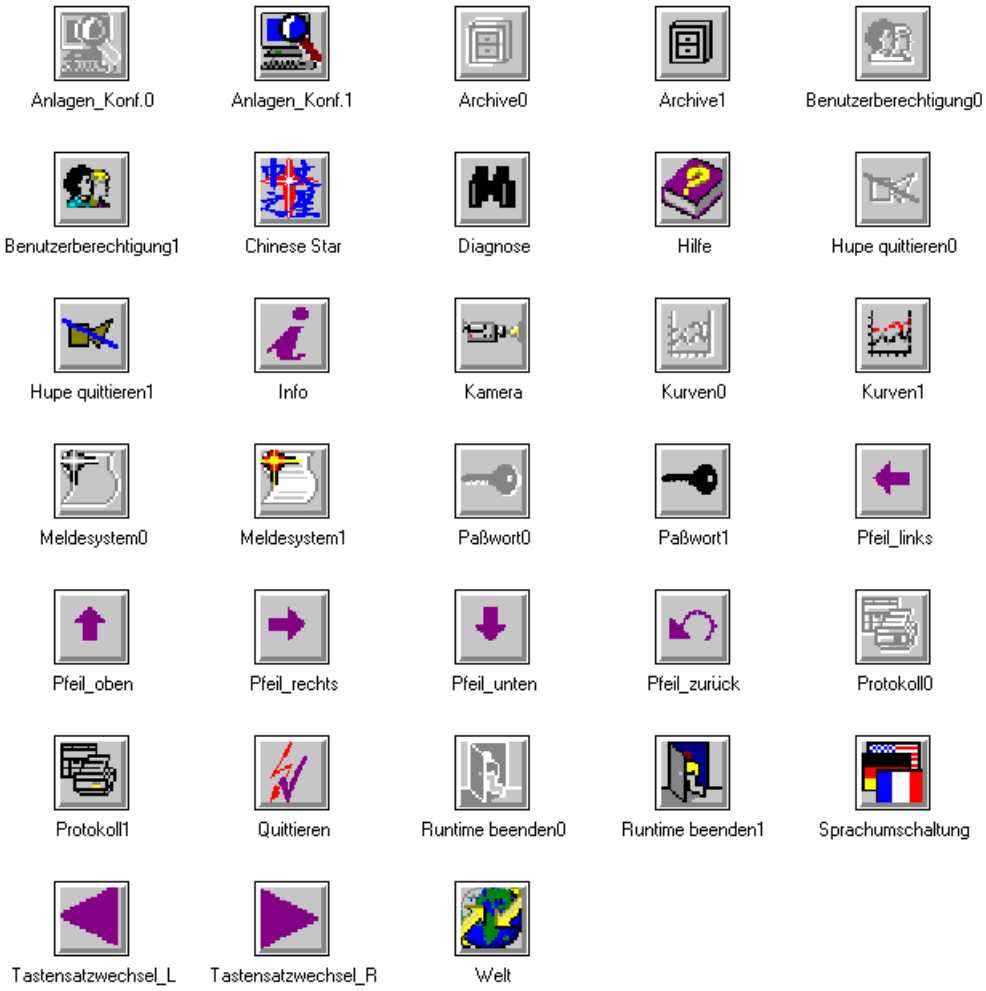
5.4.3 Bedienung

5.4.3.1 3D-Buttons



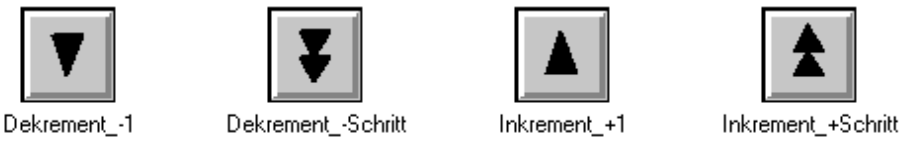
5.4.3.2 Bedienpanels

5.4.3.3 Bild-Buttons

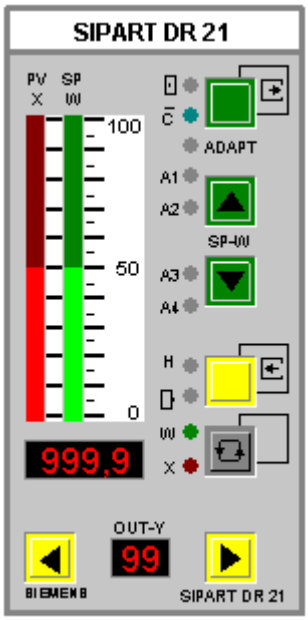


5.4.3.4 Bildnavigation

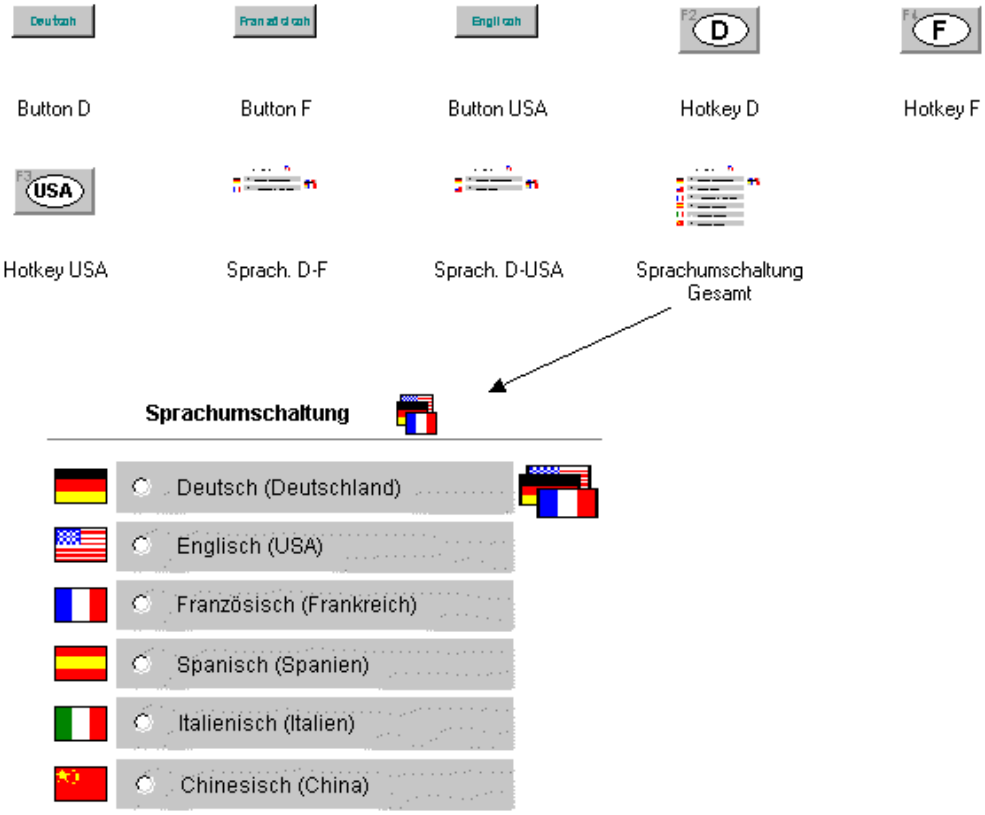
5.4.3.5 Incr./Decr.-Buttons



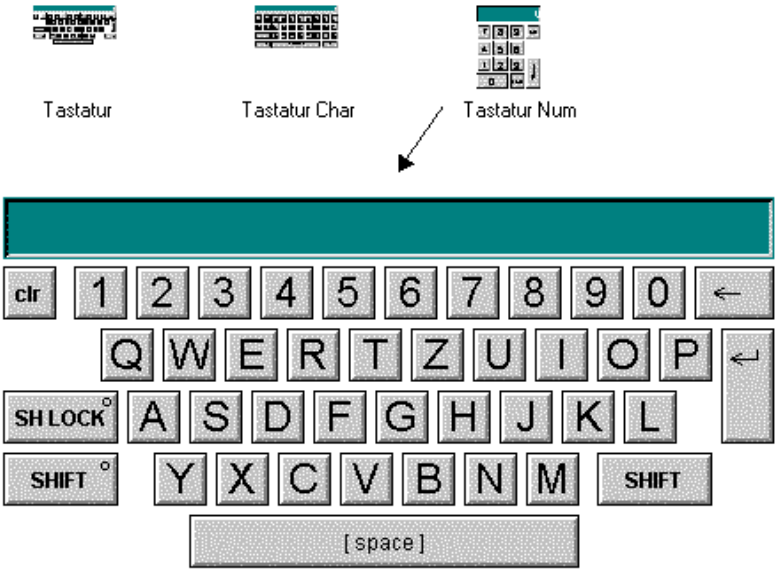
5.4.3.6 Regler



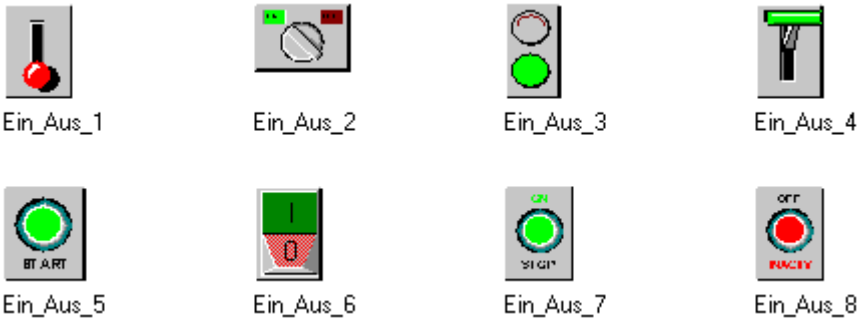
5.4.3.7 Sprachumschaltung



5.4.3.8 Tastaturen

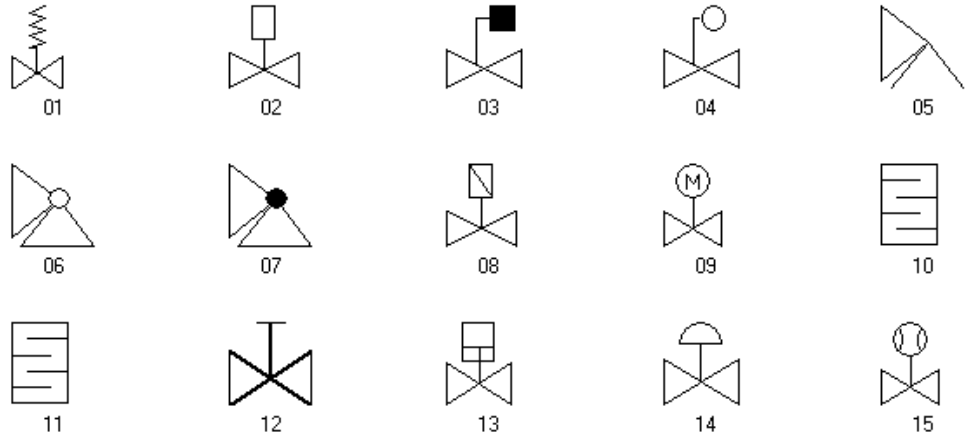


5.4.3.9 Umschalt-Buttons

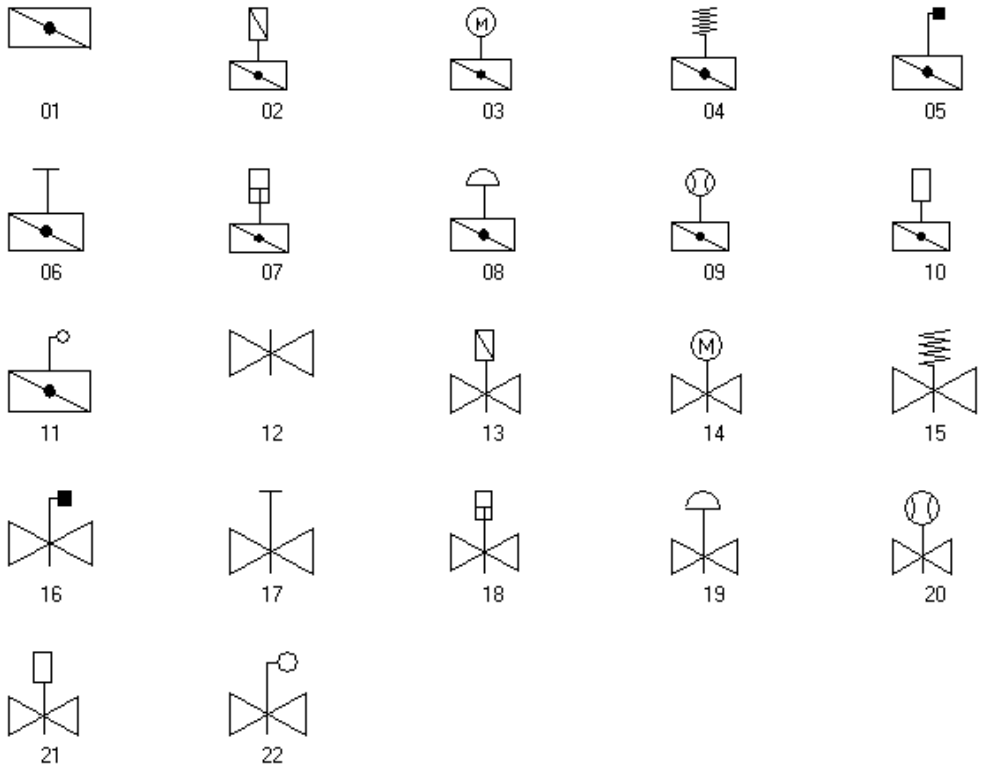


5.4.4 Symbole

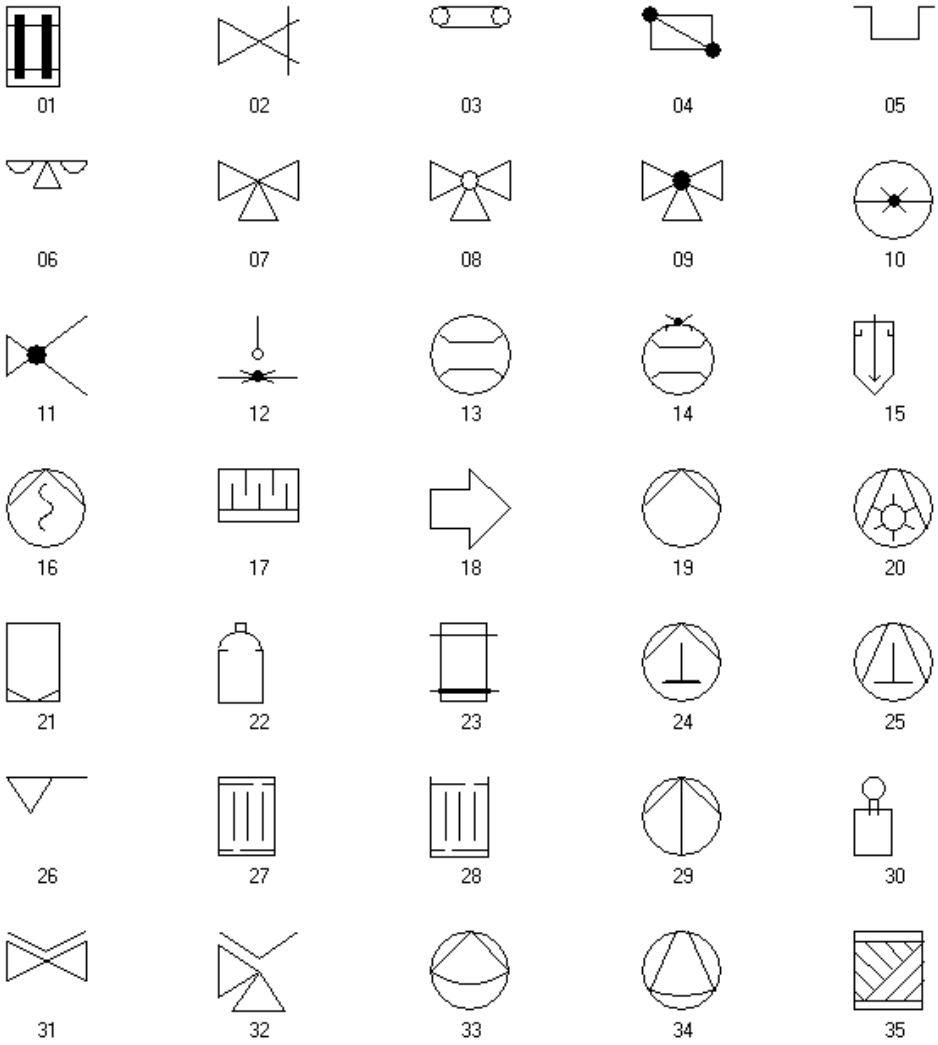
5.4.4.1 Absperrarmaturen



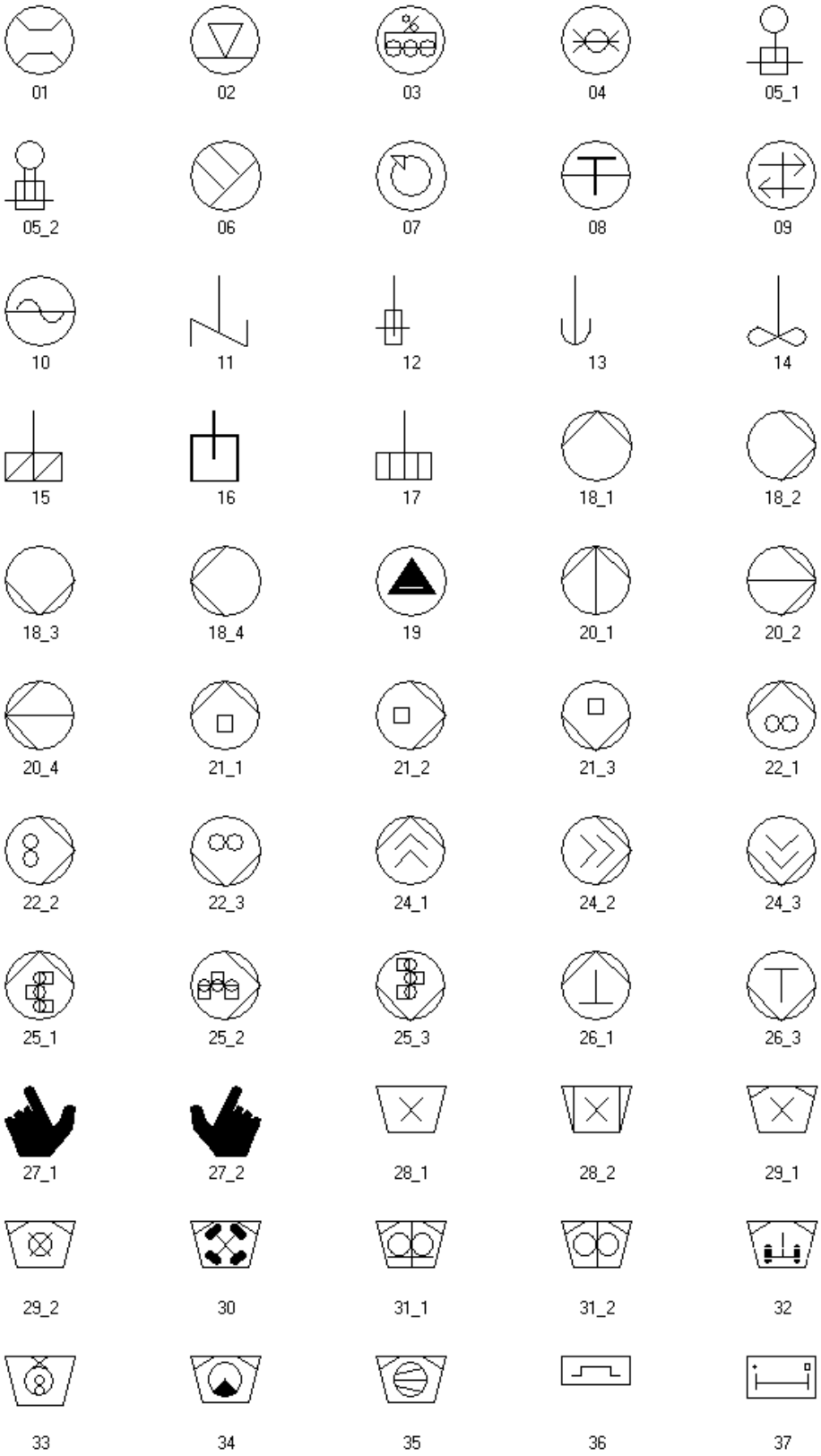
5.4.4.2 Absperrschieber



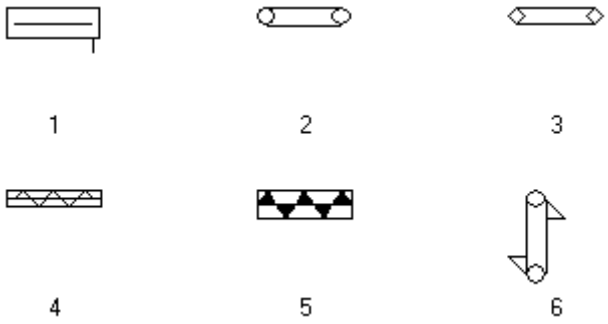
5.4.4.3 DIN 30600



5.4.4.4 E-Symbole

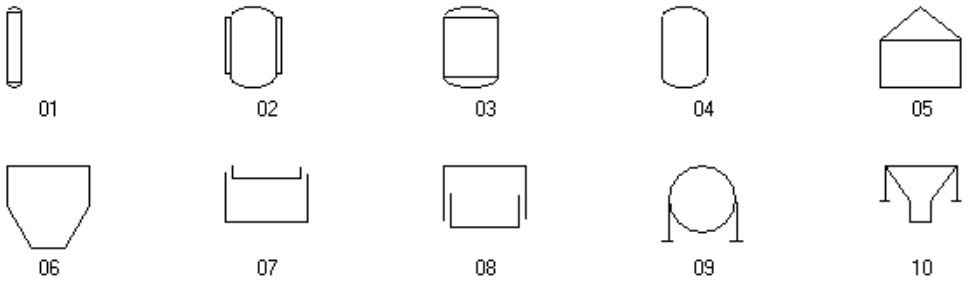


5.4.4.5 Förderbänder



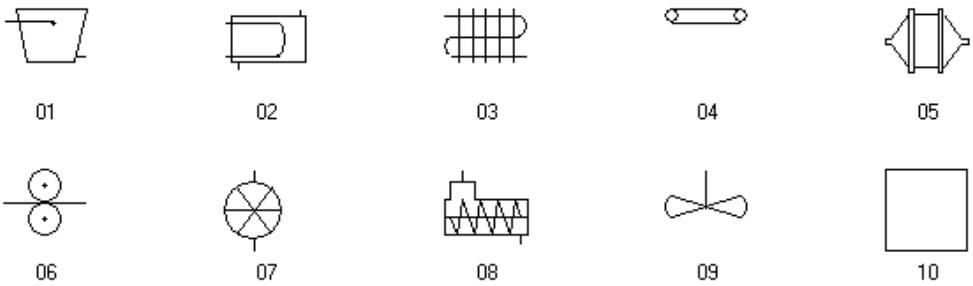
5.4.4.6 ISA-Symbole

5.4.4.6.1 Isa_s55a

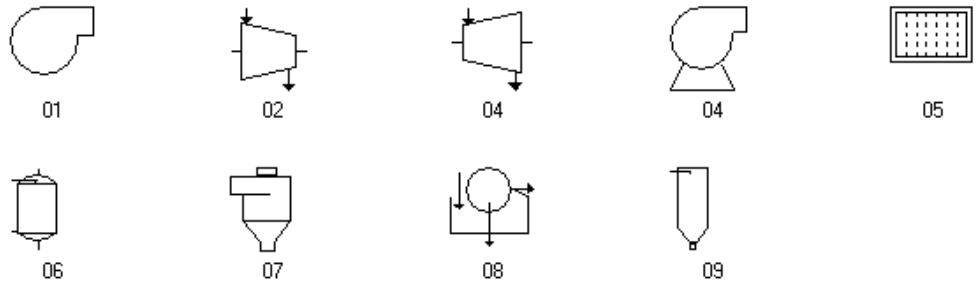


5.4.4.6.2 Isa_s55b

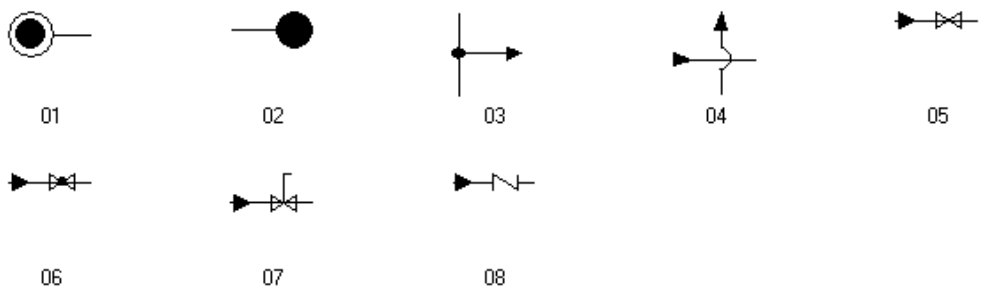
5.4.4.6.3 Isa_s55c



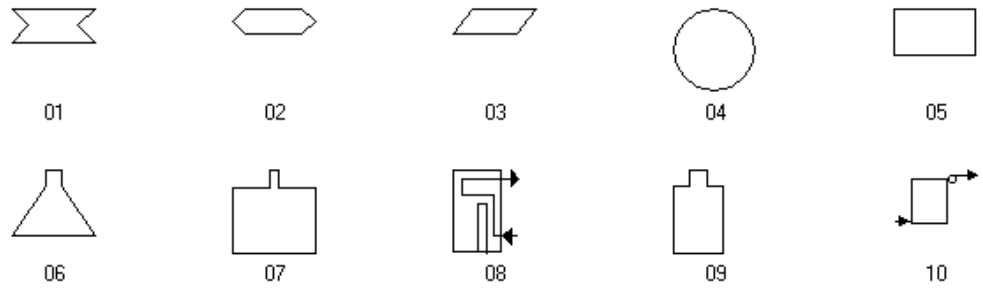
5.4.4.6.4 Isa_s55d



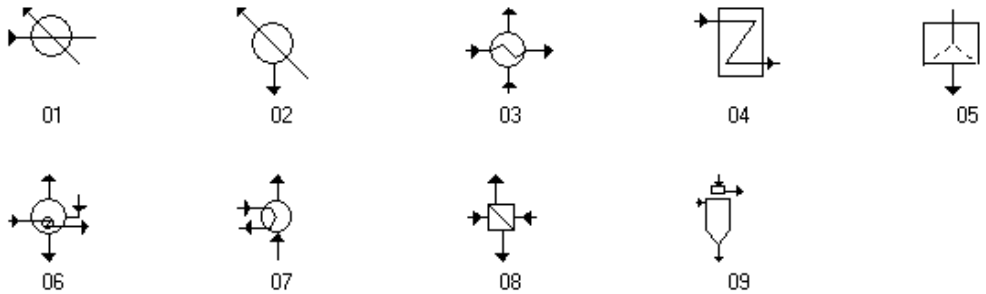
5.4.4.6.5 Isa_y32a



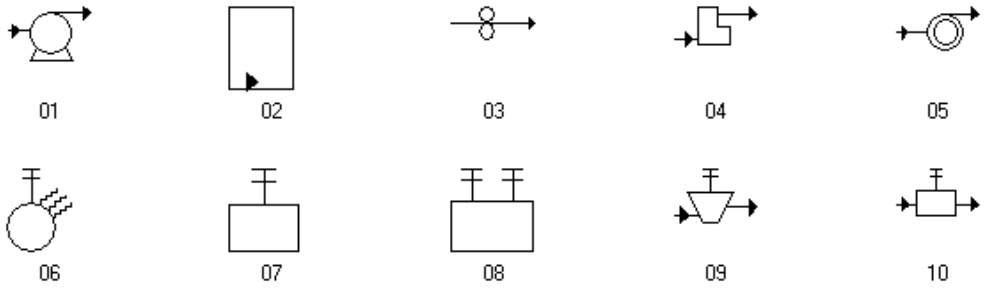
5.4.4.6.6 Isa_y32b



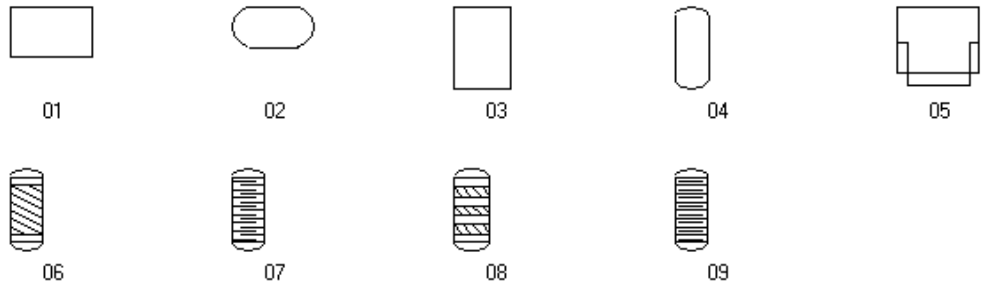
5.4.4.6.7 Isa_y32c



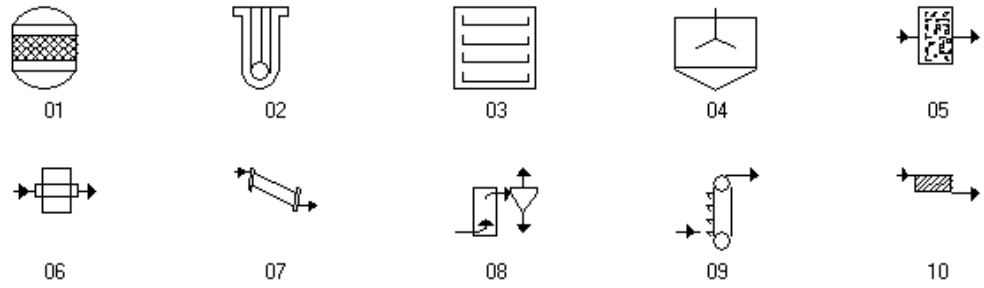
5.4.4.6.8 Isa_y32d



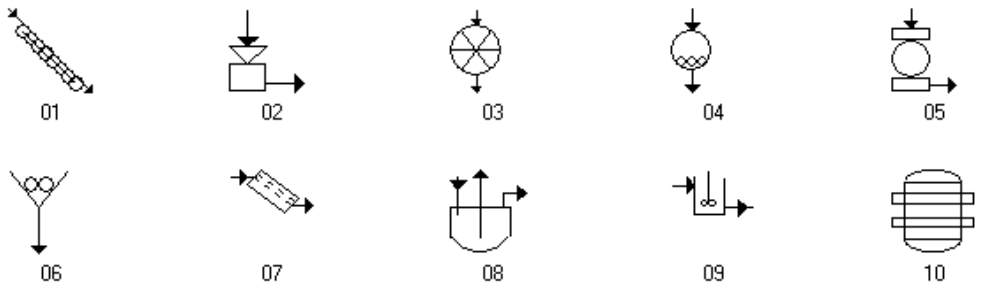
5.4.4.6.9 Isa_y32e



5.4.4.6.10 Isa_y32f

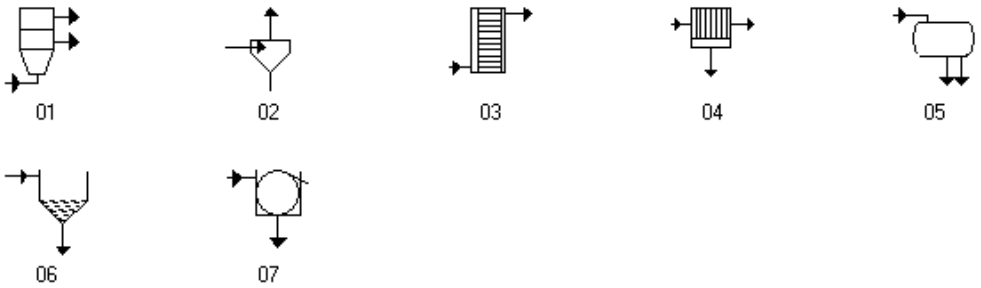


5.4.4.6.11 Isa_y32g

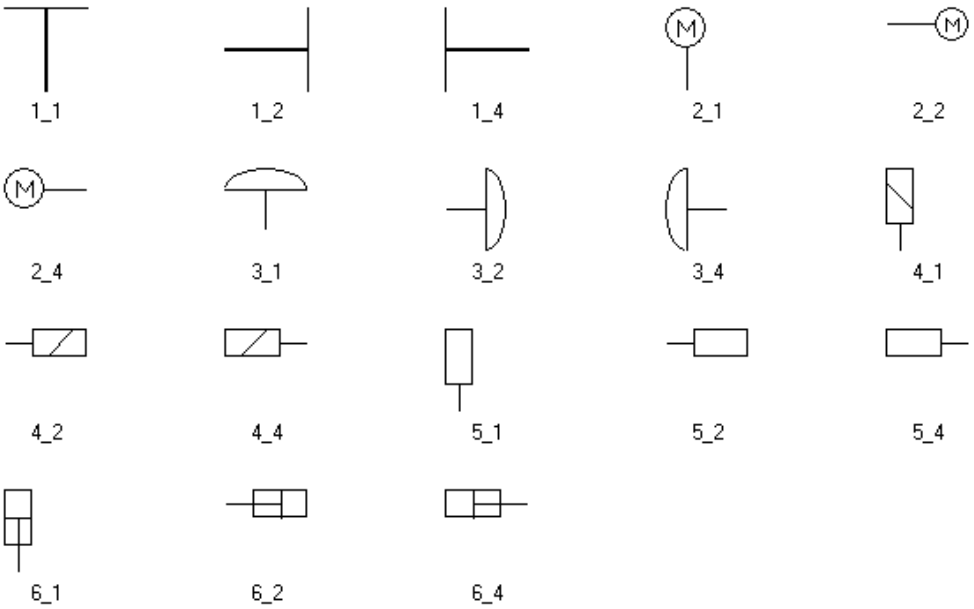


5.4.4.6.12 Isa_y32h

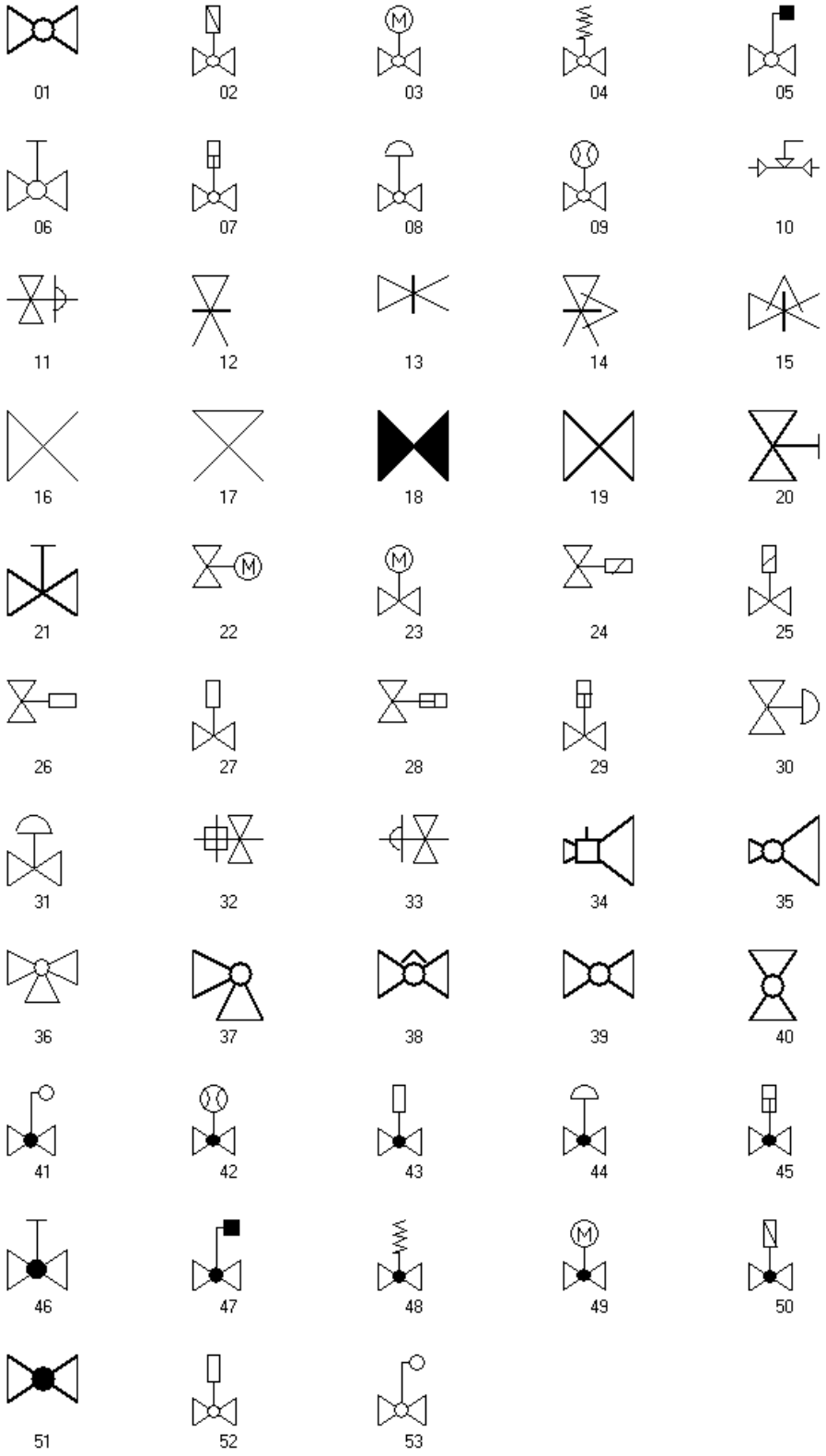
5.4.4.6.13 Isa_y32i



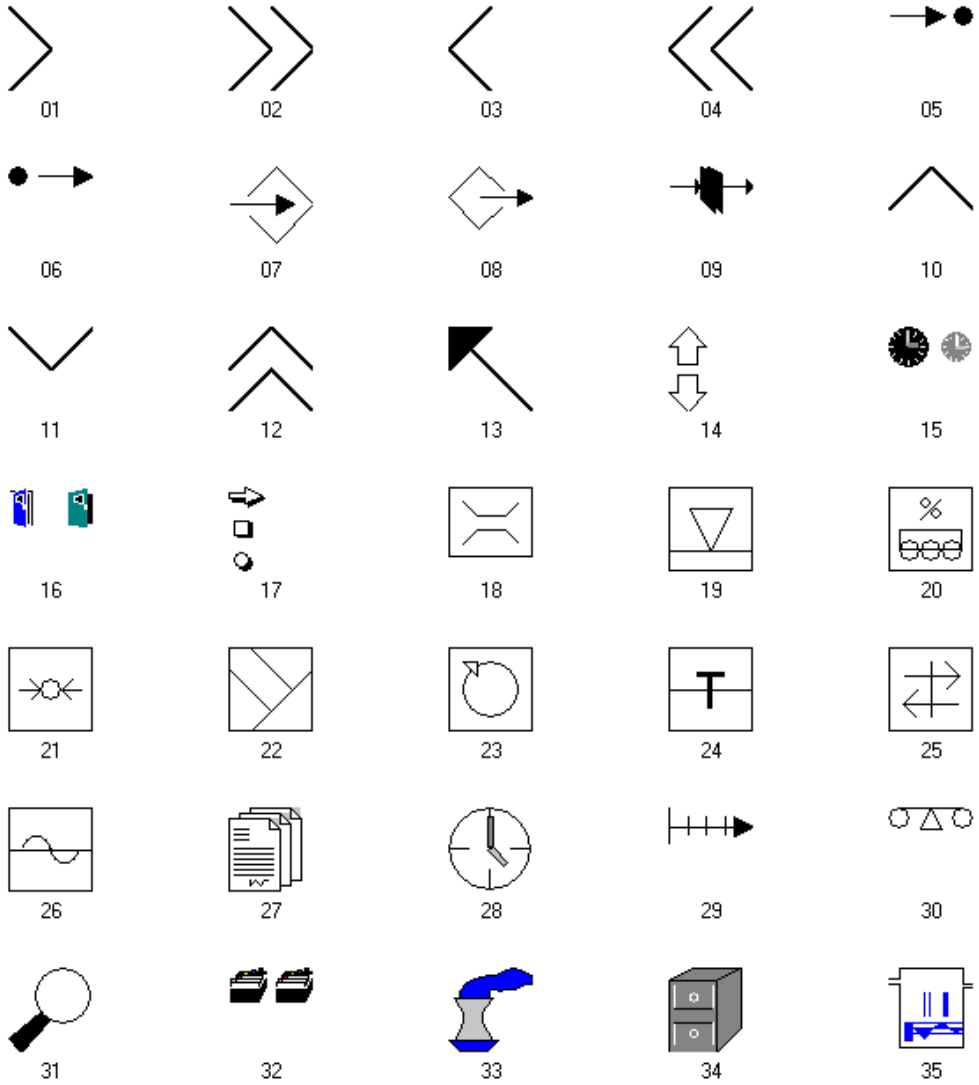
5.4.4.7 Motoren



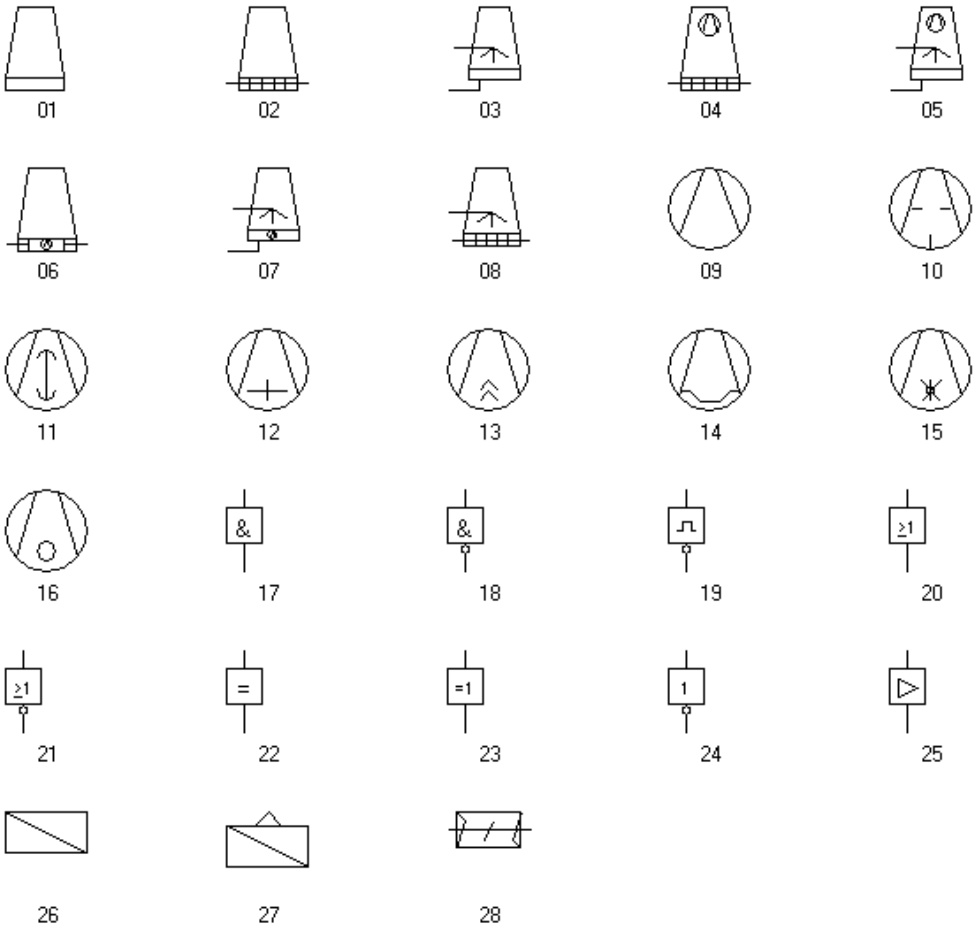
5.4.4.8 Ventile



5.4.4.9 Verschiedenes 1



5.4.4.10 Verschiedenes 2



Index

A

Abfragen
  Ereignis am Objekt 3-38
Abschalten
  Runtime-Cursor 3-82
Abwicklung 1-2, 3-1
Access
  WinCC Daten lesen 5-8
ActiveX 2-2, 2-3, 3-54, 3-105
Addition 4-31
Adressierung
  Indirekt 3-5, 3-96
Aktionen 4-13
  Aktualisierung im Bild 3-23
  Aktualisierungszyklen 3-26
  Ändern 4-12
  Bei Bildanwahl 5-3
  Editor für 4-12
  Erstellen 4-3, 4-15
  Festlegen 3-8
  In WinCC 4-4
  Übernehmen 3-64
  Verzeichnis 3-45
  Wiederverwenden 4-12
Aktualisierung
  Arten der 3-26
  Möglichkeiten der Einstellung 3-22
  Zyklen 3-17
Alarm
  Allgemeines zur Festlegung 3-19
  Im Bedienkonzept 3-14
Alphacursor 3-14, 3-82
Analogwerte
  Darstellen 3-10
  Uhrzeit 3-105
Ändern
  Anwenderzyklus 3-24
  Bildzyklus 3-29
  Fensterzyklus 3-31
  Mit Variablentrigger 3-27
  Projekteigenschaften 3-55
  Von Zuständen 3-37
Anfordern
  Daten 3-22
  Daten vom Datenmanager 3-26
Anforderungen 1-1, 3-9
Anlagenbild
  Dynamisieren 3-39
  Mauslos bedienen 3-77
Anlauf
  Automatischer 3-47
  Systemmeldungen beim 3-41
Anlegen
  Datei mit Script 4-115
  Meldestruktur 3-75
Anpassung
  Bildaktualisierung 3-23
  Daten für Variablenimport 3-71
  Rechnereigenschaft 3-55
Anschluß
  An den Prozeß 3-17
  USV 3-50
ANSI 4-3
Anwahl 3-14, 3-28, 3-30, 3-85
Anweisung
  Bedingte 4-49
Anwender 2-3, 3-9
  Anwenderdatensätze 2-3
Anwenderarchiv 3-59, 3-107
  Übernehmen 3-75
Anwenderbibliothek 3-81
Anwenderobjekt 3-98, 3-100, 3-102, 3-104
  In Bildbausteintechnik 3-93
  Übernehmen 3-62
  Wizard 3-98
Anwenderzyklus 3-24, 3-35
API 2-4, 3-65, 3-66, 3-107
Applikation 3-22
  Eigene 2-2
  Fremde einbinden 2-3
  Schließen 3-48
  Schnittstelle (API) 3-65
Applikationsfenster 2-3, 3-84, 3-88
Arbeitsbereich 3-13
archivieren 2-2
Archivierung 1-1, 3-2, 3-52
  Archivierungszeiten 3-22
  Projektieren 3-57
Aufbau 1-2
  Bedienkonzept 3-15
  Bildaufbau 3-22, 3-93
  Funktionen 4-5
  Projektierhandbuch 1-2
  Textlisten 3-69
  Variablenhaushalt 3-5
  Verbindungen 3-102
  Verbindungsaufbau 3-108
  Verzeichnisstruktur 3-20
  WinCC 2-2
Aufgaben 1-1
  Projektteam 3-57
Auflösung
  Bildschirm 3-13

Ausblenden
Von Informationen 3-10
Ausführen
Variablen Import- Export 3-68
Von Scripten 3-34
Ausgabe
Normiert 5-2
Testausgabe 4-16
Auslagern
Meldungen 3-43
Variablen 3-66
Auswahl
Bilder für Zustandsanzeige 3-60
Der Variablen 3-5
Meldungen 3-19
Trigger 3-23
Autorisierung 3-54
Autostart
Eintrag im Ordner 3-47

B

Basic 2-2, 3-106
Basic Process Control 3-15, 3-81
Basis
Basisprojekt 3-57, 3-75
Für Import 3-68
Bausteine
Bildbausteintechnik 3-93
Vorprojektiert 3-62
Bedienbar
Bedienobjekte 3-82
Funktionen mit Zugriffsschutz 3-49
Kurvenfenster 3-88
Meldefenster 3-84
Bedienkonzept 3-14
Bedienoberfläche 2-3, 3-2
Festlegen 3-9
Bedienobjekte 3-82
Bedienung 3-11, 3-14
Ereignisgesteuert 3-17
Fehlbedienung 3-10
Kurvenfenster 3-88
Mauslos 3-77
Über Funktionstasten 3-78
Über Tastatur 3-37, 3-81
Bedingte
Anweisungen 4-49
Beenden
Eingabe 3-83
Von WinCC 3-50
WinCC 3-47
Begriffe 2-5
Beispielprojekt 1-1, 3-57
Bildschirmauflösung 3-13
Sprache 3-3
Benutzer
Berechtigung 3-109
Gruppen 3-18, 3-76
Rechte 3-18
Übernahme der Rechte 3-76
Zeitzyklen benutzerdefiniert 3-24
Bereich
Arbeitsbereich 3-13
Bildschirm 3-12
Tastenbereich 3-16
Wertebereich 3-39, 4-20
Zeitbereich 3-88
Betriebssystem 2-2, 3-17, 3-50
Bibliothek
Projekt 3-53, 3-62, 3-63
Standard 3-42
Text 3-72
Bild
Aktualisierung 3-23
Anwahl 3-14, 3-59, 3-84, 3-88, 3-105
Aufbau 3-22, 3-93
Bausteintechnik 3-93
Bildfensterinhalt 3-59
Größe der 3-13
Hierarchie 3-81
Informationen im Bild 3-10
Meldebild 3-85
Namen 3-7
Objekt 3-37
Übernahme von 3-58
Volumen reduzieren 4-12
Wechsel 3-78
Zyklus 3-24
Zyklus ändern 3-29
Bildschirmauflösung 3-13
Bitmap
Übernehmen 3-60
Bitmeldeverfahren 3-19
Bitoperationen 4-36
Bussystem 3-17

C

Callback-Funktion
zur Rückmeldung des Status eines Schreibauftrags 5-68
zur Übergabe gelesener Prozeßwerte 5-68
C-API 2-4
ChnStart 5-66
Client 2-4
Coros 3-73
Cursor

Alpha 3-14, 3-82
Schalt 3-14, 3-82

D

Dateien
Im Projektverzeichnis 3-44
Im Standardverzeichnis 3-41
Daten
Ablage 3-4, 3-20
Aktualisierung 3-17
Anfordern 3-22
Datenbestand 2-4
Datenhaltung 2-3
Importieren 3-52
In der Datenbank 3-43
Projektierungsdaten 2-3
Sicherung 3-53
Trennung 3-44
Übernahme 3-57
Übernahme von S5 oder S7 3-65
Datenbank 2-3, 2-4
Abfragesprache 2-4
Änderung 3-57
Rekonstruktion 3-43
Selektion in 5-13
Sicherung 3-52
Zugriff mit Access 5-8
Zugriff mit ISQL 5-10
Zugriff mit Scope 5-11
Datenexport
Über C Aktion 5-12
DDE 2-2
Default
Einstellungen 3-3
Sprache 3-42
Trigger 3-36
Verzeichnis 3-4
Definition
C Strukturen 4-67
Dekrementieren 4-31
Diagnose 4-17
Datei für Import 3-68
Dateien 3-41
Scope 5-4
Dialog
Konfiguration 3-39, 3-60
Konfigurations 3-23
Direktverbindung 3-23, 3-39, 3-81, 3-96, 3-102
Division 4-31
Dokumentation 1-1, 1-2, 3-2
Dynamic-Wizard 3-23, 3-29, 3-39, 3-65, 3-93, 3-98
Dynamik-Dialog 3-23, 3-27, 3-39
Dynamische
Änderung der Objekteigenschaften 3-37
Instanz 3-100
Verbindung der Variablen 3-96
Dynamisierung
Anwenderobjekt 3-99
Art 3-27
Arten 3-23
Eigenschaften 3-37
Ereignisse 3-38
In WinCC 3-37
Objekte 3-39
Projektieren 3-39

E

Editor 4-12
Eigenschaft
Am Objekt Dynamisieren 3-26
Funktionskopf an der 4-6
Objekt 3-23
Eigenschaften einer Kanal-Unit
Client-Funktionalität 5-69
Diagnosemöglichkeiten 5-69
Editieren der Kanal-Properties 5-69
Eigene Lebenszeichenüberwachung 5-69
Eigene Wiederanlaufanzeige 5-69
Eigene Zyklusverwaltung 5-69
Keine Registrierung von WinCC-Variablen 5-69
Online-Registrierung logischer Verbindungen 5-69
Online-Registrierung von WinCC-Variablen 5-69
Prozeßwerte in INTEL Bytereihenfolge 5-69
Reentrant 5-69
Schreiben auf Bit-Adressen 5-69
Schreiben auf Byte-Adressen 5-69
Uhrzeit-Master 5-69
Uhrzeit-Slave 5-69
Zugriff auf Remote-Variable 5-69
Einbinden
Von OCX 3-93
Eingabe
In EA-Feld 3-82
Mittel 3-14
Normiert 5-2
Über Tastatur 3-77
Einlesen
Meldungen 3-73
Einplatz
System 3-13, 3-50
Einschränkung

Bei Onlineprojektierung 3-107
Bildname 3-7
Variablenname 3-6
Einzelplatz
System 3-57
Empfehlung
Für Aktualisierungszyklen 3-27
EN 60073 3-16
Entwicklung 2-2
Entwicklungsumgebung 3-45, 3-106, 4-3
Ereignis
Dynamisieren 3-38
Funktionskopf am 4-5
Gesteuert 3-17
Trigger 3-23, 3-34, 3-36
Ergebnis
Ausgeben 4-16
Dynamisierung 3-40
Excel 3-6, 3-20, 3-68, 3-70, 3-73, 3-75
Export 3-6, 3-43, 3-52, 3-63, 3-68
Extension
Für Aktionen 3-64
WinCC Dateien 3-44

F

Farb
Definition 3-16
Erkennung 3-10
Farben im Projekt 3-11
Farbtabelle 5-15
Fehler
Bedienfehler 3-44
Fehlermeldung 3-56
Suche 3-41, 4-12
Fenster
Diagnose 4-17
Fensterwechsel 3-84
Zyklus 3-24, 3-26, 3-31
Float 4-20
Funktionalität 1-1
Bildbausteine 3-100
Toolbar Alarm Logging 3-87
Toolbar Tag Logging 3-90
Funktionen
API 3-65
Aufbau 4-5
Bedien 3-18
Editor 4-12
Erstellen 4-13
In WinCC 4-4
Interne 3-26, 4-12
Kopf 4-5
Projekt 3-45, 4-12
Projektfunktionen übernehmen 3-76
Standard 3-41, 3-76, 4-12
Zugriffsschutz 3-49
Funktionstaste 3-77

G

Generieren
Header neu 3-59, 3-76, 3-104
Header neu erstellen 3-56
Gleitkommazahlen 4-20
Grafik
Bibliothek 3-95
Tools 3-43
Grundrechnungsarten 4-33
Gruppe
Benutzergruppe 3-18, 3-76
Benutzerrgruppen 3-59
Variablengruppe 3-5

H

Hardcopy 3-91
Hierarchie
Bedienung 3-14
Bei Anlagen 3-78
Bild 3-81
HMI 2-2, 2-5, 3-17, 3-47
Hotkey 3-78, 3-79, 3-84, 3-92

I

IF 4-50
Import 3-6, 3-52, 3-65, 3-68
Information
Ausblenden 3-10
Finden 3-4
Im Bild 3-9
Informix 2-4
Ingres 2-4
Inhalt 1-2
Projektverzeichnisse 3-44
Initialisierung einer Kanal-Unit
Parameter für den WinCC-Datenmanager 5-68

Inkrementieren 4-31
Installation
Tools 3-43
USV 3-50
WinCC 3-54
WinCC Installationspfad 3-41
Instanz 3-100
Anlegen 3-100

Interne
Funktionen 3-26, 3-41

K

Kanal
DLL 3-54, 3-66, 3-71
S7 PMC 3-108
Kommunikation 3-17, 3-59
Einfluß auf Aktualisierung 3-26
Online Ändern 3-108
Prozeß 3-22
Schnittstellen 3-54
Zwischen einzelnen Tasks 3-27
Konfiguration
Dialog 3-23, 3-28, 3-39, 3-54, 3-60, 3-98
Konvention 1-2
Konzept 2-4
Bedienung 3-11, 3-14
Datensicherung 3-50
Prototypenbilder 3-94
Kopplung
Seriell mit CP525 5-14
Kurven
Fenster Bedienung 3-84, 3-88

L

Laufwerk
Zur Datensicherung 3-52
Laufzeit 3-99, 3-104
Daten 3-45
Dynamisierung zur 3-37
Lizenzprüfungen 3-41
Log
Dateien 3-41, 3-46
Logik 4-32
Vergleich 4-32
Login 3-92, 3-109
Logoff 3-109
Logout 3-92
Lösung
Automatischer Projektanlauf 3-47
Lösungsansatz 2-2
Lösungsvorschlag 1-1
Lösungsweg 1-1

M

Maus
Aktion für Hotkey 3-79
Dynamisierung Ereignisse 3-38
Mauslose Bedienung 3-77
Mehrplatz
System 3-13, 3-50, 3-57
Melde
Meldeklasse 3-16, 3-74
Meldeliste 3-84
Meldesystem 3-16
Meldefenster 3-59, 3-84, 3-88
Meldungen 2-3
Bitmeldeverfahren 3-19
Einlesen 3-73
Farben festlegen 3-16
Meldebild 3-85
Meldedatei 3-74
Meldeverfahren 3-19
Sicherung 3-52
Systemmeldungen 3-41
Übernehmen 3-72, 3-73
Zeitfolgerichtiges Melden 3-19
Meßwert
Aktualisirung 3-17
Archivierung 3-22
Auslagern 3-43
Erfassung 3-57
Übernehmen 3-75
Modularität 2-2
Module
Einbinden 2-2
Multiclient 3-57
Multiplikation 4-31

N

Netzwerk 3-52

O

OCX 2-3, 3-54, 3-83, 3-105
Bildbaustein 3-93
Registrierung 3-43, 3-106
ODBC 2-2
OLE 2-2
Verbindungen 3-54
Online
Projektierung 3-107
Operatoren 4-31
Optionen 3-15, 3-43, 3-44, 3-81
Oracle 2-4

P

Parameter
Default 3-45
Diagnose 3-108

Für Bildnamen 3-7
Für Bildschirmauflösung 3-13
Für printf 4-17
Für Projektnamen 3-4
Für Variablennamen 3-6
Shutdown 3-51
Verbindungs 3-66
Paßwort 3-18, 3-49, 4-12
Performance 1-1, 3-2, 3-17, 3-39, 3-57
Plattform 2-2
Printf 4-16
Programm
Autostart 3-47
Eigene 3-65
Für Datenbank 3-43
Tools 3-6
Zur Datenübernahme 3-66
Zusatzprogramme 3-54
Programmierschnittstelle 2-4
Projekt
Automatischer Anlauf 3-47
Beispielprojekt 3-3
Bibliothek 3-63
Bibliothek übernehmen 3-62
Funktion erstellen 4-13
Funktionen 4-12
Kopieren 3-54
Name 3-4
Sicherung 3-52
Tips zur Realisierung 3-20
Übergreifende Funktionen 3-42
Umgebung 3-44
Verzeichnis 3-44
Wartungsfreundlich 3-25
Projektierung
Daten der 3-44
Dynamisieren 3-39
Festlegungen für 3-2
Modus 3-47
Projektierungsdaten 2-3
Prototyp 3-93, 3-94, 3-98, 3-101
Prozeß
Anschluß 3-17
Bedienung 3-11
Hierarchie der Bedienung 3-14
Kommunikation 3-22
Variablen 3-96
Prozeßdaten 2-4, 3-22
Archivieren 2-2

Q

Quittierung
Meldungen 3-84

R

Radiobox
Bedienung über Tastatur 3-83
Rechner
Einstellungen 3-48
Hotkey 3-84
Typ ändern 3-107
Verzeichnis für 3-45
Referenz
In Bildbausteinen 3-95, 3-104
Textreferenz 3-72
Registrieren
OCX 3-43, 3-106
OLE, OCX 3-54
Rekonstruktion
Datenbank 3-43
Richtungstasten 3-83, 3-85
Rückgabewert 4-5
Runtime
Bedienung im 3-82
Cursor 3-82
Dynamisierung 3-37
Einschränkung bei Onlineprojektierung 3-107
Festlegungen für 3-2

S

Schaltcursor 3-14, 3-82
Schleifen 4-49
Do while 4-49
For 4-49
While 4-49
Schnittstelle
API 3-65
Schrift
Art 3-2
Farbe 3-16
Größe 3-2, 3-13
Scope 5-4
Zugriff auf WinCC Datenbank 5-11
Scripte 3-8
Ausführen 3-34
Editoren für 4-12
Entwicklungsumgebung 4-3
Für Wizard 3-42
Sprache 4-1
Syntax 4-3
Shutdown 3-50
Sicherung 3-44
Konzept für 3-50
WinCC Daten 3-52
SmartTools 3-6, 3-43, 3-54, 3-66
Sonderzeichen 3-7

Speicher 4-42
Sprache 3-72
Übernehmen 3-72
SQL 2-3, 2-4
Datenbank 3-48
Programmierung 3-66
Tools 3-43
Standard
C 4-3
Einstellungen 3-23, 3-24
Funktionen 3-53, 4-12
Inhalt WinCC Standardverzeichnis 3-41
Installaitonspfad 3-41
Tasten 3-80
Zyklen 3-27
Start
WinCC automatisch 3-47
String 4-42
Struktur
Bildnamen 3-7
Datenbank 3-57
Für Datenablage 3-20
Für Meldungen anlegen 3-75
Variablen 3-5
WinCC Projektverzeichnis 3-44
WinCC Systemverzeichnis 3-41
Strukturen 4-67
Subtraktion 4-31
Sybase 2-3, 3-48
Symbol
Ablage der 3-42
Exportieren 3-63
Für Dynamisierungsart 3-37
Übernahmen 3-60
Syntax 4-3
System
Automatischer Anlauf 3-47
Belastung 3-22, 3-27, 3-36
Betriessystem 3-17
Einplatz 3-13
Einzelplatz 3-57
Layout 3-53
Mehrplatz 3-13, 3-57
Meldungen 3-41
Module 2-2
Plattform 2-2
Software 3-54
Umgebung 3-41
Variablen 3-6, 3-59

T

Tabellenfenster 3-59, 3-88
Task
Taskleiste 3-48
Taskwechsel 3-26
Tastatur
Bedienung 3-14, 3-37, 3-77
Funktionstasten 3-80
Hotkey 3-91
Tasten
Bedientasten 3-59
Funktionstasten 3-77
Hotkey 3-79
Im Kurvenfenster 3-88
Im Meldefenster 3-85
Projektierte 3-72
Tastenbelegung 3-83
Tastenbereich 3-12, 3-16
Tasteneinstellungen 3-82
Tastenkombination 3-47
Template
Bilder 3-101
Meldefenster 3-85
Text
Darstellung 3-11
Erkennung am Bildschirm 3-11
Listen 3-69
Mehrsprachige 3-72
Textlibrary 3-109
Textlisten 3-65, 3-83
Toolbar
Alarm Logging 3-84, 3-87
Tag Logging 3-88
Tools
Datenbank 3-52
Grafik 3-43
OCX 3-54
Sprachen 3-72
SQL 3-43
Variablen Import- Export 3-6, 3-66
WinCC 3-43
Tooltip 3-13
Transaktionsgesichert 2-3
Trigger
Bildaktualisierung 3-23
Im Dynamik-Dialog 3-32
Variablen 3-23
Variablentrigger 3-27
Zeitgesteuert 3-35

Ü

Übersichtsbilder 3-14
Umlaufpuffer
Für Bilder 3-14
Umwandlung
Einplatz - Mehrplatz 3-57

UNIX 2-4
USV 3-50

V

Variablen
- Anbindung 3-39
- Archivvariablen 3-89
- Aufbau der Exportierten Listen 3-69
- Auslagern 3-66
- C-Variablen 4-19
- Festlegen 3-5
- Import- Export 3-52
- In Scripten 3-6
- Informationen im Bild Darstellen 3-95
- Meldung über fehlende 3-41
- Name 4-20
- Referenzen in Bildern 3-59
- Simulation 3-46
- Trigger 3-23, 3-24, 3-27, 3-36
- Typen 4-20
- Übernehmen 3-65
- Von S5 / S7 übernehmen 3-65
- Zulässige Namen 3-6

VDE 0199 3-16
Verbindung
- Aufbauen 3-108
- Indirekt 3-93
- Logische 3-65
- Mit Prozeßvariablen 3-103
- Mit Variablen 3-96
- Neue 3-93
- Verbindungsliste 3-70

Verzeichnis
- Daten im WinCC Verzeichnis 3-42
- Für Autostart 3-47
- Projektbibliothek 3-63
- Projektverzeichnis 3-4
- Projektverzeichnis von WinCC 3-44
- Struktur von WinCC 3-20, 3-41
- Tools für WinCC 3-66

Visual
- Basic 2-2, 3-106
- C++ 2-2, 3-106

Visualisieren 2-2
Visualisierung
- Station 3-22

Voraussetzung 1-2
Vorlage
- Für eigene Projekte 3-3

Vorschriften
- Bei der Datenübernahme 3-71

W

While 4-49
WinCC
- Aktion übernehmen 3-64
- Aktionen 3-8
- Aktualisierungszyklus 3-23
- Alamierung 3-19
- API 3-65
- Aufbau 2-2, 2-3
- Automatischer Anlauf 3-47
- Bedienkonzept 3-14
- Bedienoberfläche 3-9
- Beenden 3-50
- Datensicherung 3-52
- Defaulteinstellung 3-3
- Dynamisierung 3-37
- Logdateien 3-41
- Projekt kopieren 3-54
- Projektname 3-4
- Projektumgebung 3-44
- Scripte 3-8
- Standartverzeichnis 3-41
- Systemumgebung 3-41
- Tools 3-43, 3-54
- Variablen übernehmen 3-65
- Variablenhaushalt 3-5
- Version 1.10 3-74
- Verzeichnisstruktur 3-44

Windows 2-2
- Windows NT 3-17

Wizard
- Coros Meldungen einlesen 3-73
- Dateien im Standardverzeichnis 3-42
- In Anwenderobjekten 3-98
- S5 / S7 Variablen einlesen 3-65
- Script übernehmen 3-108
- Zur Bildaktualisierung 3-23
- Zur Objektdynamisierung 3-39

Wrebuild 3-43
Wunload 3-43

Z

Zahl
- Ganzzahlen 4-20
- Gleitkommazahl 4-20

Zeichen
- Folge 4-42
- Zulässige bei Bildnamen 3-7
- Zulässige bei Variablennamen 3-6
- Zulässige im Projektnamen 3-4

Zeiger
- In C 4-41

Zeigerinstrument 3-10, 3-99

Zeitfolgerichtiges
Melden 3-19
Zeittrigger 3-34
Zeitzyklus 3-23, 3-26, 3-35
Zielgruppe 4-1
Zugriff
Auf Datenbank 3-52, 3-65
Rechte 3-59
Zugriffsschutz 3-14, 3-18, 3-49
Übernahme der Rechte 3-76
Zuordnungslisten 3-65
Zustand
Änderung 3-19
Beim Verlassen 3-47
Darstellung von Zuständen 3-10
Zustandsanzeige 3-60
Zykluszeit 3-26