ISaGRAF

Version 3.4

BENUTZERHANDBUCH

CJ INTERNATIONAL

Die in diesem Dokument enthaltenen Informationen können ohne Vorankündigung geändert werden und stellen keine Verpflichtung seitens CJ International dar. Die Software, die in diesem Dokument beschrieben wird und die Informationen einschließt, die in jeder Datenbank enthalten sind, wird unter einer Lizenz- und Geheimhaltungsvereinbarung zur Verfügung gestellt und darf nur in Übereinstimmung mit den Bedingungen dieser Vereinbarung benutzt oder kopiert werden. Es verstößt gegen das Gesetz, die Software zu kopieren, wenn nicht ausdrücklich in der Lizenz- und Geheimhaltungsvereinbarung erlaubt. Ohne die ausdrückliche schriftliche Erlaubnis von CJ International darf kein Teil dieses Handbuchs zu irgendeinem Zweck, in irgendeiner Form, mit irgendwelchen Mitteln, elektronisch oder mechanisch, mittels Fotokopie, durch Aufzeichnung oder mit Informationsspeicherungs- und Informationswiedergewinnungssystemen reproduziert oder übertragen werden.

Gedruckt in Frankreich von CJ International.
3 Rue Hector Berlioz
F-38600 FONTAINE
Phone: 33 (0)4 76 26 87 30
Fax: 33 (0)4 76 26 87 39

Aus dem Englischen übersetzt von I. Stephenson.

Inhaltsverzeichnis

B. SPRACHREFERENZEN B-175

A. Benutzerhandbuch

A.1. Inbetriebnahme

Dieses Kapitel beschreibt die Installation der ISaGRAF Workstation. Anhand einer kurzen Beispielanwendung wird ein Überblick über die hauptsächlichen ISaGRAF Funktionalitäten vermittelt und die sofortige Inbetriebnahme von ISaGRAF ermöglicht.

A.1.1. ISaGRAF installieren

Dieses Kapitel beschreibt, wie die ISaGRAF Workstation installiert wird und welche Konfiguration für die Entwicklung einer Anwendung erforderlich ist.

Hardware- und Software-Anforderungen

Für die Installation der ISaGRAF Workstation kommen alle Geräte in Frage, die Windows Version 3.1 unterstützen. Für die Entwicklung von Anwendungen möchten wir Ihnen jedoch die folgende Hardware-Konfiguration empfehlen:

- PC mit einem 80486 Mikroprozessor oder höher
  (ein Pentium Prozessor wird empfohlen)
- 8 Megabytes konventioneller und Erweiterungsspeicher
  (16 Megabytes werden empfohlen)
- Diskettenlaufwerk 3.5 Zoll (1.44 Megabyte)
- Festplatte mit minimal 20 Megabytes verfügbarem Speicherplatz
- VGA- oder SVGA-Grafikkarte und kompatibler Monitor
- Maus (für die grafischen Entwicklungswerkzeuge)
- Paralleler LPT1-Anschluß (für das Dongle)

Bevor Sie die ISaGRAF Workstation installieren, sollte die folgende Software auf Ihrem System vorhanden sein:

- Windows Version 3.1 im erweiterten 386-Modus
- Windows 95
- Windows NT Version 3.51 oder 4.00

Installationsprogramm benutzen

Die Installation der ISaGRAF Workstation erfolgt mit dem ISaGRAF Installationsprogramm INSTALL. Dieses Programm kopiert die ISaGRAF Software von der ISaGRAF CD-ROM oder den ISaGRAF Disketten auf Ihre Festplatte, fügt die Gruppe "ISaGRAF" in das Fenster des Programm-Managers ein und erstellt eine Initialisierungsdatei namens "ISA.ini" im installierten Unterverzeichnis **EXE**.

INSTALL ist ein Windows Programm, das vom Windows Programm-Manager aus oder mit dem Run-Befehl des Startmenüs von Windows 95 gestartet wird. Die Installation von ISaGRAF verläuft wie folgt:

- ISaGRAF CD-ROM oder Diskette #1 in das entsprechende Laufwerk eingeben.

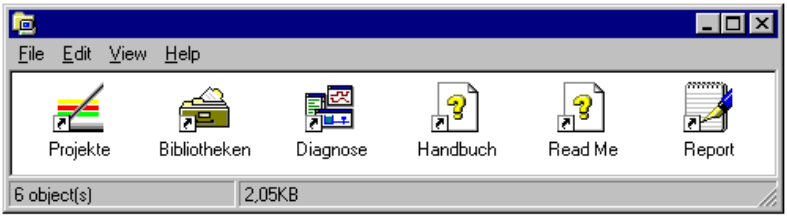
• Im Programm-Manager oder dem Start-Menü "SETUP.EXE" im Wurzelverzeichnis der CD-ROM einleiten (bei Disketten "A:\INSTALL.EXE" eingeben).
• Folgen Sie den Programmanweisungen, um die Installation zu vervollständigen. Die ISaGRAF Workstation sollte in einem neuen Verzeichnis installiert werden, um Konflikte mit Dateien anderer ISaGRAF Versionen zu vermeiden.

INSTALL fragt nun, ob Sie die folgenden Komponenten installieren möchten:

• Ausführbare ISaGRAF Programme
• Online-Informationen und Hilfsdateien
• ISaGRAF Standardbibliotheken
• ISaGRAF Beispielanwendungen

Bei einer ersten Installation der ISaGRAF Workstation sollten alle Komponenten miteinbezogen werden. Sollte dies nicht der Fall sein, können fehlende Komponenten durch die erneute Installation von ISaGRAF auch später noch hinzugefügt werden.

Der Vorgabename für das ISaGRAF Hauptverzeichnis ist "\ISAWIN". Auf diese Weise kann ISaGRAF für Windows problemlos auf einer Festplatte installiert werden, auf der schon eine Version von ISaGRAF für MS-DOS vorhanden ist. Siehe auch Abschnitt "ISaGRAF Verzeichnisse" im Kapitel "Fortgeschrittene Techniken" für ausführliche Informationen über die ISaGRAF Verzeichnisarchitektur auf der Festplatte. Wenn alle ISaGRAF Dateien kopiert sind, erscheint die folgende Gruppe im Fenster des Windows Programm-Managers:



Die hauptsächlichen ISaGRAF Symbole sind:

Projekte:..... Projektverwaltung
Bibliotheken:..... Verwaltung der Bibliotheken
Diagnose:..... Diagnosewerkzeug für den Endbenutzer
Handbuch:..... Online-Informationen zu ISaGRAF
Read Me: ..... Informationen zur neuen ISaGRAF Version
Report: ..... Standardformat für den Debugging-Bericht

Wenn ein Problem angetroffen wird, können die relevanten Daten in das Standardformat für den Debugging-Bericht eingegeben werden. Sichern Sie die Datei (Datei-Menü/ Sichern unter) und senden Sie Ihren Bericht direkt an CJ International per Fax oder E-Mail.

Systemdateien aktualisieren

Nach beendeter Installation muß die Systemdatei CONFIG.SYS aktualisiert werden, bevor der Computer erneut gestartet wird. Es ist nicht nötig, den Pfadnamen des ISaGRAF Verzeichnisses in die PATH-Umgebungsvariable einzufügen, da ISaGRAF keine MS-DOS Umgebungsvariablen benutzt. Folgende Anweisungen können der CONFIG.SYS Datei jedoch hinzugefügt werden:

```
files=20
buffers=20
```

Die ISaGRAF Workstation benutzt einen seriellen Anschluß für die Kommunikation mit der ISaGRAF Zielsteuerung. Für ISaGRAF wird der Anschluß COM1 vorgegebenen. Wenn Ihre Maus ebenfalls einen seriellen Anschluß benutzt, sollte sie besser an COM2 angeschlossen werden, so daß die COM1-Vorgabe für alle neuen ISaGRAF Anwendungen gültig ist.

Nach der Aktualisierung der CONFIG.SYS Datei muß der Computer erneut gestartet werden, damit allen Änderungen Rechnung getragen wird.

⇒ **Wichtig für Windows NT Anwender:**
Wenn die Workstation unter Windows NT 3.51 oder 4.00 benutzt wird, muß dem [WS001] Abschnitt der ISA.ini Datei im Verzeichnis \ISAWIN\EXE die folgende Zeile hinzugefügt werden:

```
[WS001]
NT=1
Isa=C:\ISAWIN
IsaExe=C:\ISAWIN\EXE
IsaApl=C:\ISAWIN\APL1
IsaTmp=C:\ISAWIN\TMP
```

Dies ist für die RS-Kommunikation unbedingt erforderlich.

Das Dongle

Ein Dongle schützt die ISaGRAF Software gegen illegale Kopien. Selbst wenn das Dongle nicht angeschlossen ist, sind die meisten Funktionen der ISaGRAF Workstation weiterhin verfügbar. Das Dongle definiert auch die Optionen der ISaGRAF Workstation und die maximale Größe der entwickelten Anwendungen. Ohne das Dongle oder bei fehlerhaftem Anschluß sind bestimmte Funktionen der ISaGRAF Workstation nicht zugreifbar. Dieses Verhalten ist NORMAL. Um festzustellen, ob das Dongle richtig angeschlossen ist, wählen Sie "**Info...**" im "**Hilfe**"-Menü der ISaGRAF Fenster. Der Name der verfügbaren ISaGRAF Workstation wird dann angezeigt.

Das Dongle kann an einen beliebigen Parallelanschluß Ihres Computers angeschlossen werden. Wenn Ihr Gerät über mehrere Parallelanschlüsse verfügt, ist es besser, das Dongle und den Drucker an verschiedene Anschlüsse anzuschließen. Bei manchen PC/Drucker-Konfigurationen wird das Dongle möglicherweise nicht erkannt, wenn sein Ausgang an einen "Offline"-Drucker angeschlossen ist. In diesem Fall muß der Drucker entfernt oder "online" geschaltet und die ISaGRAF Workstation erneut gestartet werden.

Für die **ISaGRAF 32** Workstation wird kein Dongle benötigt.

⇒ **Wichtig für Windows-NT Anwender:**
Auf Windows-NT Systemen muß der Sentinel/Rainbow™ Treiber installiert werden, damit das Dongle sichtbar wird. Hierzu wird eine spezielle Diskette geliefert.

A.1.2. Online-Informationen benutzen

Die mit der ISaGRAF Workstation installierten Online-Informationen geben Auskunft über folgende Themen:
- ISaGRAF Sprachreferenzen
- Komplettes Benutzerhandbuch (für alle ISaGRAF Werkzeuge)
- Technische Datenblätter für die Bibliothekselemente

Online-Informationen können jederzeit im "**Hilfe**"-Menü eines beliebigen ISaGRAF Fensters angefordert werden.

A.1.3. Eine Beispielanwendung

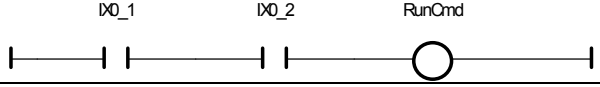
In diesem Kapitel wird schrittweise beschrieben, welche Operationen zur Erstellung, Entwicklung, Generierung und zum Testen einer einfachen, aber vollständigen Multisprachen-Anwendung erforderlich sind.

Es folgen die kompletten Spezifizierungen dieser Anwendung, in der KOP- und AS-Darstellungen miteinander kombiniert werden:

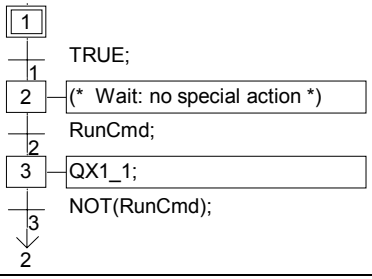
Boolesche Variablen:

IX0_1, IX0_2:	Eingangsvariablen
RunCmd:	Interner "Run/Stop"-Befehl
QX1_1:	Ausgangsvariable: Prozeßstatus

Program Command: Zyklischer Anfangsabschnitt - KOP-Sprache
Bewertet den internen "Run/Stop"-Befehl



Program RunStop: Sequentieller Abschnitt - AS-Sprache
Prozeßsteuerung



Start ***ISaGRAF Workstation starten***

Um die ISaGRAF Workstation zu starten, leiten Sie den Befehl "Projekte" in der "ISaGRAF" Gruppe vom Windows Start-Menü aus ein.

Projekterstellung

Erstellen Sie das Projekt (genannt "RunStop") mit Hilfe des Befehls "Neu" im "Datei"-Menü oder durch Klicken auf die Schaltfläche "Neu". Ein Dialogfeld öffnet sich:

Geben Sie den Projektnamen ein: "**RunStop**"
Wählen Sie die E/A-Konfiguration: "**Sim_Boo**"
Klicken Sie auf die Schaltfläche "**OK**".
Das Projekt ist nun erstellt.

Projekt öffnen

Zur Definition der Projektprogramme wird das Fenster des ISaGRAF Programm-Managers geöffnet. Wählen Sie den Befehl "Bearbeiten" im Fenster des Projektmanagers oder doppelklicken Sie mit der Maus auf den Projektnamen. Sie können auch auf die Schaltfläche "Bearbeiten" klicken.

Erstellung der Programme

Das Programm-Manager-Fenster ist jetzt offen und leer (keine Programme definiert). Das erste Programm wird mit dem Befehl "Neu" im "Datei"-Menü oder mit der Schaltfläche "Neu" erstellt. Ein Dialogfeld wird angezeigt:

Geben Sie den Progammnamen ein: "**Command**".
Wählen Sie die Sprache: "**Schneller KOP**".
Wählen Sie den Abschnitt "**Zyklusanfang**".
Klicken Sie "**OK**", um das Programm zu erstellen.

Diese Operation wird für das zweite Programm wiederholt:
Benutzen Sie den Befehl "Neu" im "Datei"-Menü oder die Schaltfläche "Neu". Ein Dialogfeld wird angezeigt:

Geben Sie den Progammnamen ein: "**RunStop**".
Wählen Sie die Sprache: "**AS**".
Wählen Sie den Abschnitt "**Sequentiell**".
Klicken Sie "**OK**", um das Programm zu erstellen.

Die Programme sind jetzt erstellt. Sie erscheinen im Fenster des Programm-Managers.

Die Variablen deklarieren

Bevor die Programminformationen eingegeben werden können, muß die interne Variable, die in der Programmierung benutzt werden soll, deklariert werden. Dies erfolgt unter Anwendung des Befehls "Datenverzeichnis" im "Datei"-Menü oder durch Klicken auf die Schaltfläche "Datenverzeichnis". E/A-Variablen werden automatisch bei der Projekterstellung deklariert.

Das Datenverzeichnis-Fenster wird angezeigt. Über das "Datei"-Menü, das Untermenü "Andere", das Untermenü "Globale Variablen" und den Befehl "Boolesche" wählen Sie das Datenverzeichnis der globalen Booleschen Variablen.

Sie können hierzu auch die Schaltflächen "Globale Objekte" und "Boolesch" benutzen.

Der Befehl "Neu" im "Datei"-Menü wird benutzt, um neue Boolesche Variablen zu erstellen. Sie können auch die Schaltfläche "Objekte hinzufügen" benutzen. Im geöffneten Dialogfeld geben Sie die Beschreibung der internen Variablen ein:

Name:	**RunCmd**
Kommentar:	**Run/Stop-Befehl: intern**
Attribut:	Wählen Sie das Attribut "**Intern**"

Klicken Sie auf die Schaltfläche "**Eingeben**": Die Variable ist erstellt.
Klicken sie auf "**Abbrechen**", um das Dialogfeld zu schließen.

Jetzt beenden Sie den Datenverzeichnis-Editor und sichern die Änderungen: "**Datei**"-Menü /Befehl "**Beenden**". Klicken Sie "**JA**", um die Änderungen zu sichern.

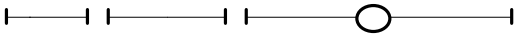
Bearbeitung des Schnellen KOP-Programms

Um die Bearbeitung des KOP-Programms "Command" zu starten, doppelklicken Sie auf seinen Namen im Fenster des Programm-Managers oder benutzen Sie die Schaltfläche "Bearbeiten".

Das Fenster des Schnellen ISaGRAF KOP-Editors wird jetzt angezeigt. Bringen Sie das Fenster auf seine maximale Größe, um das Arbeitsfeld zu erweitern.

F2 F3 Drücken Sie die Tasten F2 und F3:

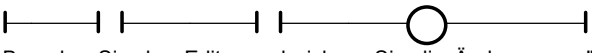
(* *)



Ordnen Sie den KOP-Symbolen Variablen zu: Bewegen Sie den Cursor mit den Pfeiltasten auf die Symbole und drücken Sie die Eingabe-Taste. Das Dialogfeld für die Variablenauswahl wird angezeigt.
Für den ersten Kontakt geben Sie IX0_1 in das Dialogfeld ein, dann drücken Sie die Eingabe-Taste.
Für den zweiten Kontakt geben Sie IX0_2 in das Dialogfeld ein, dann Eingabe.
Für die Spule geben Sie RunCmd ein, dann Eingabe.

Das Programm ist fertig. Hier das Ergebnis:

IX0_1 IX0_2 RunCmd



Beenden Sie den Editor und sichern Sie die Änderungen: "**Datei**"-Menü / Befehl "**Beenden**". Klicken Sie "**JA**", um die Änderungen zu sichern.

Bearbeitung des AS-Programms

Um die Bearbeitung des AS-Programms "RunStop" einzuleiten, doppelklicken Sie auf seinen Namen im Fenster des Programm-Managers oder benutzen Sie die Schaltfläche "Bearbeiten".

Das Fenster des AS-Editors wird jetzt angezeigt. Bringen Sie das Fenster auf seine maximale Größe, um das Arbeitsfeld zu erweitern:

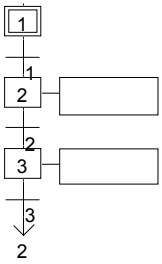
↓ Der Initialisierungsschritt existiert bereits und ist angewählt. Drücken Sie die Pfeiltaste "Nach unten", um die leere Zelle nach dem Initialisierungsschritt (0,1) anzuwählen.

F4 F3 Drücken Sie F4 und dann F3, um einen Schritt und eine Transition einzufügen.

F4 F3 Drücken Sie F4 und F3, um wiederum einen Schritt und eine Transition einzufügen.

F5 Drücken Sie F5, um einen Sprung zu einem Schritt einzufügen, und wählen Sie GS2 als Sprungziel.

Das Netzwerk ist jetzt fertig. Klicken Sie auf die "Zoom"-Schaltfläche in der Befehlsleiste, um die Zellen zu vergrößern und Platz für die Anzeige der Ebene-2-Anweisungen zu schaffen. Das Netzwerk sieht so aus:



↵ Für die Programmierung der Transition Nr. 2 wählen Sie die Transition mit Hilfe der Pfeiltasten aus und drücken die "Eingabe"-Taste. Das Fenster für die Programmierung der Ebene 2 öffnet sich. Geben Sie die Ebene-2-Programmierung der Transition Nr. 2 ein:

RunCmd;

^TAB Drücken Sie die Tasten "Strg + Tab", um zum AS-Netzwerk zurückzukehren und verlagern Sie die Auswahl auf Schritt Nr. 3. Drücken Sie die "Eingabe"-Taste, um den Text der Ebene 2 einzugeben:

QX1_1;

Wiederholen Sie diese Operationen für die Transition Nr. 3:

Not (RunCmd);

^F4 Drücken Sie die Tasten "Strg + F4", um das Fenster der Ebene 2 zu schließen.

Das AS-Programm ist jetzt fertig. Beenden Sie den Editor ("**Datei**"-Menü, Befehl "**Beenden**)" und klicken Sie auf "**JA**", um die Änderungen zu sichern.

Entwicklung des Anwendungscodes

Benutzen Sie den Befehl "**Code generieren**" im Menü "**Codierung**" im Fenster des Programm-Managers oder die entsprechende Schaltfläche in der Befehlsleiste, um den Anwendungscode zu entwickeln.

Nach erfolgreicher Code-Entwicklung wird ein Dialogfeld angezeigt, das Sie fragt, ob Sie die Code-Entwicklung **jetzt** beenden wollen oder ob Sie **fortfahren** möchten. Klicken Sie auf die Schaltfläche "**Beenden**".

Simulation

Benutzen Sie den Befehl "**Simulieren**" im Menü "**Debugging**" im Fenster des Programm-Managers oder die entsprechende Schaltfläche in der Befehlsleiste, um den ISaGRAF Kernelsimulator zu starten.

Wenn das Simulatorfenster angezeigt wird, kann die Anwendung getestet werden. In diesem Beispiel müssen die Eingänge 1 und 2 (grüne Schaltflächen) gedrückt werden, um den Prozeß zu starten (Ausgang rotes LED).

Schließen Sie das Debugger-Fenster, um die Simulation zu beenden: "**Datei**"-Menü / Befehl "**Beenden** ".

A.2. Projekte verwalten

Um den ISaGRAF Projektmanager zu starten, doppelklicken Sie mit der Maus auf das Symbol "Projekte" in der ISaGRAF Gruppe. Das Fenster der "Projektverwaltung" wird angezeigt.
Ein Projekt entspricht einer Zyklusschleife in der Zielsteuerung. Das obere Fenster enthält die Liste der bestehenden Projekte. Der Projektkopf des ausgewählten Projekts wird im unteren Fenster angezeigt.

Fenstergröße verändern

Klicken Sie einfach auf die Trennlinie (Splitter) zwischen Projektliste und Projektkopf, um die Größe der entsprechenden Fenster zu verändern. Das Projektkopf-Fenster kann nicht ganz versteckt werden. Es enthält mindestens eine Textzeile.

Trennlinien einfügen

Eine Trennlinie kann vor dem Namen eines beliebigen Projekts eingefügt werden. Auf diese Weise können Projekte, die zu einer bestimmten Anwendung gehören, in der Liste gruppiert werden. Benutzen Sie den Befehl "**Bearbeiten / Trennlinie**", um eine Trennlinie vor dem ausgewählten Projekt einzufügen oder zu löschen.

Projekte in der Liste verschieben

Um ein Projekt in der Liste zu verschieben, muß es zuerst ausgewählt werden (es hebt sich dann farblich ab). Klicken Sie nun auf seinen Namen und ziehen Sie es an eine neue Stelle in der Liste. Beim Verschieben des Projekts zeigt ein kleiner Pfeil am linken Rand an, wo das Projekt eingefügt wird. Sie können auch die Befehle "**Verschieben**" im Menü "**Bearbeiten**" benutzen, um das ausgewählte Projekt zeilenweise zu verschieben. Wenn eine Trennlinie vor dem ausgewählten Projekt eingefügt wurde, so wird diese zusammen mit dem Projekt verschoben.

A.2.1. Projekte erstellen und bearbeiten

Die Befehle im Menü des Projektmanagers dienen der Erstellung neuer Projekte, der Bearbeitung dieser Projekte und der Verwaltung bestehender Projekte.

Ein neues Projekt erstellen

Um ein neues Projekt zu erstellen, muß zuerst sein Name eingegeben werden. Man erhält ein leeres Projekt, das noch keine Objekte enthält. Diesem neu erstellten Projekt kann eine in der Bibliothek definierte E/A-Konfiguration zugeordnet werden. Nach Auswahl einer Konfiguration konfiguriert ISaGRAF den E/A-Anschluß automatisch und deklariert die entsprechenden E/A-Variablen im neuen Datenverzeichnis des Projekts. Bei der Erstellung oder Umbenennung eines Projekts müssen die folgenden Regeln beachtet werden:

- Der Name muß sich auf **8** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Die weiteren Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Nach der Erstellung eines Projekts benutzen Sie den Befehl "**Bearbeiten / Projektkommentar**", um einen Text einzugeben, der dann mit dem Projektnamen in der Liste angezeigt wird.

Den Projektkopf bearbeiten

Der Befehl "**Projekt / Projektkopf bearbeiten**" wird für die Bearbeitung des Projektkopfes benutzt. Dieses Dokument dient der genauen Beschreibung eines Projekts und unterscheidet es von den anderen Projekten in der Liste. Im Projektkopf können außerdem laufende Notizen zur Projektentwicklung erfaßt werden.

Projektbearbeitung

Der Befehl "**Datei / Öffnen**" öffnet das Fenster des Programm-Managers für das ausgewählte Projekt. Von diesem Fenster aus können alle Projektkomponenten (Programme, Anwendungsparameter...) verwaltet werden. Man kann auch auf den Namen eines Projekts doppelklicken, um es zu bearbeiten.

Projektprotokoll

"Das ISaGRAF System speichert alle Änderungen eines Projektkomponenten in einer Protokolldatei. Jede einzelne Änderung wird im Projektprotokoll mit Titel, Datum und Uhrzeit gekennzeichnet. In der Protokolldatei werden die neuesten **500** Änderungen zusammengefaßt. Jedes Projekt besitzt seine eigene Protokolldatei. Das Projektprotokoll ergänzt die mit den Programmen des Projekts verknüpften "Programmprotokoll"-Dateien. Mit dem Befehl "**Projekt / Projektprotokoll**" kann das Protokoll eines ausgewählten Projekts eingesehen oder ausgedruckt werden. Der Benutzer kann eins oder mehrere Elemente in der Liste auswählen und die folgenden Schaltflächen benutzen:

OK..... schließt das Fenster
Drucken..... sendet den Inhalt der Liste zum Drucker
Info..... zeigt Hilfe über dieses Dialogfeld an
[löscht] Auswahl . nimmt die ausgewählten Zeilen aus der Liste
[löscht] Alles ..... löscht die ganze Liste
Suchen..... sucht eine Zeichenkette in der Liste

Das Textfeld oberhalb der Schaltfläche "**Suchen**" dient der Eingabe einer zu suchenden Zeichenkette, wobei zwischen Groß- und Kleinschreibung nicht unterschieden wird. Wenn das Ende der Liste erreicht ist, wird die Suche am Anfang der Liste und bis zum Ausgangspunkt fortgesetzt.

Eine komplette Dokumentation ausdrucken

Der Befehl "**Projekt / Drucken**" dient dem Aufbau und Ausdruck einer kompletten Dokumentation für das ausgewählte Projekt, in der alle Projektkomponenten (Programme, Variablen, Parameter...) zusammengefaßt sind. Wenn eine spezielle (unvollständige) Dokumentation gewünscht wird, kann ein Inhaltverzeichnis definiert werden.

Datenschutz durch Paßwörter

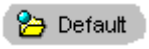
Mit dem Befehl "**Projekt / Paßwort bestimmen**" kann der Benutzer ein Paßwort zum Schutz der Werkzeuge und Daten eines ausgewählten Projekts bestimmen. Siehe Abschnitt "**Paßwort**" am Ende des ersten Teils dieses Handbuchs für weitere Informationen zu Paßwortebenen und Datenschutz. Paßwörter beziehen sich nur auf das ausgewählte Projekt. Auf andere Projekte oder die ISaGRAF Bibliotheken nehmen sie keinen Einfluß.

A.2.2. Mit mehreren Projektgruppen arbeiten

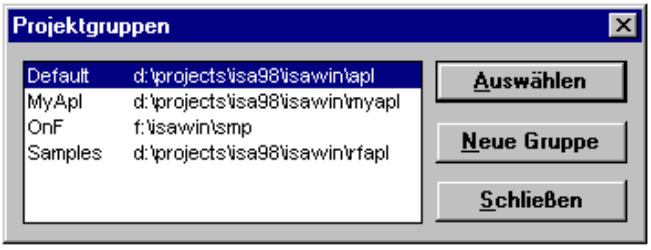
Ein ISaGRAF Projekt entspricht einem Verzeichnis auf der Festplatte, in dem alle Projektdateien gespeichert werden.
Eine "Projektgruppe" entspricht einer Liste von Projektverzeichnissen, die zusammen unter einem Wurzelverzeichnis gruppiert sind. Eine Projektgruppe wird durch einen Namen gekennzeichnet. Als Vorgabe erstellt ISaGRAF zwei Projektgruppen:

"**Default**"	in "\ISAWIN\APL"	Ihr Arbeitsbereich
"**Samples**"	in "\ISAWIN\SMP"	Beispielanwendungen, die mit der ISaGRAF Workstation geliefert werden.

Der Name der momentan ausgewählten Projektgruppe erscheint in der Befehlsleiste neben der Schaltfläche, die zur Auswahl einer Projektgruppe benutzt wird:



Sie können auch den Befehl "**Datei / Projektgruppe auswählen**" einleiten, um eine bestehende Gruppe auszuwählen oder eine neue Gruppe zu erstellen. Das folgende Dialogfeld wird angezeigt:



Wählen Sie eine Gruppe in der Liste und drücken Sie "**Auswählen**", um sie in der Liste der Projektverwaltung zu aktivieren. Sie können auch auf einen Namen doppelklicken, um die betreffende Gruppe auszuwählen. Benutzen Sie den Befehl "**Neue Gruppe**", um eine neue Gruppe zu erstellen. Dieser Befehl kann entweder benutzt werden, um einem bestehenden Verzeichnis einen Gruppennamen zuzuweisen, oder um eine neue Gruppe mit einem neuen Verzeichnis zu erstellen.
Wenn andere ISaGRAF Fenster (Programm-Manager, Editoren...) offen sind, kann keine Gruppe ausgewählt oder erstellt werden.

A.2.3. Optionen

Mit den Befehlen im Menü "**Optionen**" kann die Befehlsleiste angezeigt oder versteckt, die Schriftart für die Texte bestimmt und der automatische Schließmodus für den Projektmanager aktiviert werden. Die gewählte Schriftart gilt sowohl für die Anzeige des Projektkopfes, als auch für alle anderen ISaGRAF Texteditoren.
Wenn die Option "**Projektmanager offen lassen**" deaktiviert wird, schließt sich das Fenster des Projektmanagers automatisch, wenn ein Projekt eingegeben wird.

A.2.4. Werkzeuge

Die Befehle des "**Werkzeug**"-Menüs werden benutzt, um andere ISaGRAF Anwendungen zu starten. Der Befehl "**Werkzeuge / Projekte archivieren**" startet den ISaGRAF Archivmanager, um Projekte zu sichern und wiederherzustellen. Der Befehl "**Werkzeuge / Gemeinsame Daten archivieren**" wird benutzt, um Dateien, die von allen Projekten benutzt werden (wie z.B. allgemeine Definitionen) zu sichern und wiederherzustellen.
Der Befehl "**Werkzeuge / Bibliotheken**" startet den ISaGRAF Bibliotheksmanager in einem separaten Fenster.
Der Befehl "**Werkzeuge / AWL-Programm importieren**" kann benutzt werden, um eine Projekt zu importieren, das als einzelnes AWL-Programm in einer Textdatei beschrieben wurde (gemäß des PLC Open Dateiaustausch-Formats).

A.3. Programme verwalten

Das Fenster des Programm-Managers enthält die Anwendungsprogramme (auch Module oder Programmiereinheiten genannt). Die Menüs werden benutzt, um die Projektarchitektur zu erstellen und um Editoren, Compiler und Debugger zu starten. Bei der Entwicklung von Anwendungen ist dieses Fenster der Kern der Workstation. Das Fenster des Programm-Managers wird mit dem Befehl "**Öffnen**" im Fenster des Projektmanagers geöffnet.

A.3.1. Die Komponenten eines Projekts

Die Komponenten eines Projekts sind dessen **Programme**. Ein Programm ist eine logische Einheit, die einen Teil der Steuerungsausführung beschreibt. Globale Variablen (z.B. E/A-Variablen) können von allen Anwendungsprogrammen benutzt werden, lokale Variablen hingegen nur von einem Programm. Die Programme sind in einer **hierarchischen Baumstruktur** mit verschiedenen **logischen Abschnitten** angeordnet. In diesem Fenster erscheinen die Programme mit ihren Verknüpfungen. Die "**Hauptprogramme**" erscheinen links in der Baumstruktur.

Hauptprogramme

Die Hauptprogramme erscheinen links in der Baumstruktur. Die Hauptprogramme der ersten drei Abschnitte sind immer aktiv und werden in einem Zyklus (Scan) in der folgenden Reihenfolge ausgeführt:

- (Lesen der Eingänge)
- Ausführung der Hauptprogramme des Abschnitts **ANFANG**
- Ausführung der Hauptprogramme des Abschnitts **SEQUENTIELL**
- Ausführung der Hauptprogramme des Abschnitts **ENDE**
- (Aktualisierung der Ausgänge)

Die Programme der Abschnitte "**Anfang**" und "**Ende**" beschreiben zyklische Operationen und sind zeitunabhängig. Die Programme des Abschnitts "**Sequentiell**" beschreiben sequentielle Operationen, in denen grundlegende Operationen explizit von Zeitvariablen bestimmt werden. Die Hauptprogramme des Abschnitts "**Anfang**" werden systematisch am Anfang eines jeden Zyklus ausgeführt. Die Hauptprogramme des Abschnitts "**Ende**" werden systematisch am Ende eines jeden Zyklus ausgeführt. Die Hauptprogramme des Abschnitts "**Sequentiell**" werden gemäß der **AS**- oder **FD**-Regeln ausgeführt und müssen in der **AS**- oder **FD**-Sprache geschrieben werden, im Gegensatz zu den Programmen der zyklischen Abschnitte, die nicht in der **AS**- oder **FD**-Sprache geschrieben werden können. Ein beliebiges Programm eines beliebigen Abschnitts kann ein oder mehrere **Unterprogramme** besitzen.

Funktionen und Funktionsbausteine

Die Programme des Abschnitts "**Funktionen**" können von allen Programmen aller Projektabschnitte aufgerufen werden. Eine Funktion ist ein Algorithmus, der einen Ausgangswert aus mehreren Eingangswerten erarbeitet. Ein Funktions-Algorithmus verarbeitet ausschließlich flüchtige Variablen, die von einem zum anderen Aufruf gelöscht werden. Folglich können die Funktionen keine Funktionsbausteine

aufrufen. Ein Programm des Abschnitts "**Funktionen**" kann nicht in der **AS**- oder **FD**-Sprache geschrieben werden.
Im Gegensatz zu den Funktionen ist den "**Funktionsbausteinen**" ein Algorithmus zugeordnet, der Eingangswerte mit versteckten statischen Daten verarbeitet. Diese Daten werden vom System bei jeder neuen Benutzung des Funktionsbausteins kopiert (instanziiert). Die Programme des Abschnitts "**Funktionsbausteine**" können von allen Programmen aller Projektabschnitte aufgerufen werden. Sie können nicht in der **AS**- oder **FD**-Sprache programmiert werden.

Unterprogramme

Unterprogramme sind Funktionen, die einem (AS-, FD- oder anderen) Vaterprogramm untergeordnet sind. Ein Unterprogramm kann nur von seinem Vaterprogramm ausgeführt (aufgerufen) werden. Jedes Programm eines jeden Abschnitts kann eins oder mehrere Unterprogramme besitzen. Unterprogramme können in allen Sprachen, außer in **AS** oder **FD**, geschrieben werden.

AS-Sohnprogramme und FD-Unterprogramme

Ein **AS-Sohnprogramm** ist ein paralleles Programm, das von seinem Vaterprogramm gestartet oder gestoppt werden kann. Sowohl das Vaterprogramm, als auch sein AS-Sohnprogramm müssen beide in der **AS**-Sprache geschrieben werden.
Wenn ein Vaterprogramm ein **AS**-Sohnprogramm startet, setzt es eine **AS-Zugriffsberechtigung** in jeden seiner Initialisierungsschritte. Wenn ein Vaterprogramm ein **AS**-Sohnprogramm stoppt, entfernt es alle existierenden Zugriffsberechtigungen aus seinen Schritten.
Jedes **FD**-Programm des sequentiellen Abschnitts kann andere **FD**-Programme steuern. Während der Ausführung eines **FD**-Unterprogramms ist das FD-Vaterprogramm blockiert (wartet). Es ist nicht möglich, gleichzeitige Operationen in einem FD-Vaterprogramm und einem seiner FD-Unterprogramme auszuführen.

Verknüpfungen von Programmen und Unterprogrammen

In der hierarchischen Baumstruktur wird die Verknüpfung von Unterprogrammen und Sohnprogrammen mit ihren Vaterprogrammen durch Linien dargestellt. Die Verknüpfungslinie eines **AS**-Programms mit einem **AS**-Sohnprogramm endet in einem Pfeil. Man sollte nicht vergessen, daß eine solche Verknüpfung **parallele** Operationen darstellt.

Programmiersprachen

Jedes Programm hat seine eigene **Programmiersprache**, die bei der Programmerstellung gewählt wird und später nicht mehr geändert werden kann. **FBS**-Diagramme können allerdings Abschnitte in **KOP** enthalten und **KOP**-Diagramme Funktionsbaustein-Aufrufe einschließen. Die verfügbaren grafischen Sprachen sind: **AS** (Ablaufsprache), **FD** (Flußdiagramm), **FBS** (Funktionsbaustein-Sprache) und **KOP** (Kontaktplan). Verfügbare Textsprachen sind: **ST** (Strukturierter Text) und **AWL** (Anweisungsliste). Die Sprachen **AS** und **FD** sind den Haupt- und Sohnprogrammen des sequentiellen Abschnitts vorbehalten. Die Sprache eines Programms erscheint als Symbol neben dem Programmnamen im Fenster des Programm-Managers. Folgende Sprachsymbole werden benutzt:

 AS ..... Ablaufsprache

FD ..... Flußdiagramm
FBS ..... Funktionsbaustein-Sprache
KOP..... Kontaktplan (Eingabe mit dem Schnellen KOP-Editor)
ST..... Strukturierter Text
AWL..... Anweisungsliste

A.3.2. Programmbearbeitung

Mit den Befehlen im "**Datei**"-Menü können Programme erstellt, aktualisiert und geändert werden. Außerdem starten sie Editoren, mit denen der Inhalt der Anwendungsprogramme eingegeben werden kann.

Ein neues Programm erstellen

Mit dem Befehl "**Neu**" im "**Datei**"-Menü können Haupt-, Sohn- oder Unterprogramme in jedem Abschnitt erstellt werden.
Zuerst wird der Name des neuen Programms eingegeben, wobei folgende Regeln zu beachten sind:

- Der Name muß sich auf **8** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Die weiteren Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Als nächstes wird die Programmiersprache für das neue Programm gewählt:
AS ..... Ablaufsprache
FD ..... Flußdiagramm
FBS ..... Funktionsbaustein-Sprache (kann Abschnitte in KOP enthalten)
KOP ..... Kontaktplan (Eingabe mit dem Schnellen KOP-Editor)
ST ..... Strukturierter Text
AWL ..... Anweisungsliste

Zuletzt wird ein Ausführungsstil für das Programm gewählt:
Anfang..... Hauptebene des Abschnitts "Anfang"
Sequentiell ..... Hauptebene des Abschnitts "Sequentiell"
Ende..... Hauptebene des Abschnitts "Ende"
Funktion ..... Im Abschnitt "Funktionen"
F-Baustein..... Im Abschnitt "Funktionsbausteine"
Sohn von... ..... AS-Sohn- oder FD-Unterprogramm oder Unterprogramm eines bestehenden Programms

Wenn Sie einen der ersten fünf Abschnitte wählen, wird das Programm in die Hauptebene einer der folgenden Abschnitte gesetzt: **Anfang**, **Ende**, **Sequentiell**, **Funktionen** oder **Funktionsbausteine**. Der letzte Abschnitt kennzeichnet, ob das neue Programm ein **AS**-Sohnprogramm, ein **FD**-Unterprogramm oder ein Unterprogramm ist. Vergessen Sie nicht, daß ein sequentielles Hauptprogramm in den Sprachen **AS** oder **FD** geschrieben werden muß und daß die **AS**- und **FD**-Sprachen nicht für zyklische Programme und deren Unterprogramme benutzt werden können.

Programmkommentare eingeben

Unter ISaGRAF kann jedes Programm eines Projekts mit einem Kommentartext verbunden werden, der dann in einer kleineren Schriftgröße neben dem Programmnamen angezeigt wird. Benutzen Sie den Befehl "**Datei / Programmkommentar**", um einen Kommentar für das ausgewählte Programm einzugeben oder zu ändern.

Den Inhalt eines Programms bearbeiten

Dieser Befehl ermöglicht die Eingabe von Programmänderungen. Hierzu wird ein Editor verwendet, der von der für dieses Programm gewählten Sprache abhängig ist. Die Programmbearbeitung erfolgt in einzelnen Fenstern, so daß mehrere Programme in parallelen Fenstern bearbeitet werden können. Wenn ein Programm ausgewählt ist (sich farblich abhebt), kann seine Bearbeitung auch durch Drücken der **EINGABE**-Taste eingeleitet werden. Man kann auch einfach mit der Maus auf den Namen eines Programms doppelklicken, um dessen Bearbeitung aufzunehmen.

Bearbeitung der "Programmprotokoll"-Datei

Jedes Programm ist mit einer **Protokolldatei** verknüpft. Diese Textdatei beschreibt alle Änderungen, die im Laufe der Entwicklung eines Programms vorgenommen wurden. Die Programmprotokoll-Datei kann bearbeitet, frei modifiziert und jederzeit ausgedruckt werden. Wenn der Editor für die Änderungen am Quellcode eines Programms beendet wird, öffnet sich automatisch ein Fenster, in dem Notizen für das Programmprotokoll eingegeben werden können. Diese Notizen werden mit Eingabedatum und -uhrzeit in die Programmprotokoll-Datei aufgenommen.

Das Datenverzeichnis der Variablen

Der Befehl "**Datei / Datenverzeichnis**" startet den Editor des Datenverzeichnisses, in dem alle Projektvariablen deklariert werden. Variablen können global (allen Programmen des Projekts bekannt) oder programm-lokal sein. Der Datenverzeichnis-Editor kann auch zum Deklarieren von **Definitionen** benutzt werden, die dazu dienen, Wörter oder Ausdrücke im Quellcode eines Programms zu ersetzen.

Parameter von Funktionen, Unterprogrammen und Funktionsbausteinen

Mit dem Befehl "**Datei / Parameter**" können die Aufruf- und Returnparameter einer ausgewählten Funktion oder eines ausgewählten Unterprogramms oder Funktionsbausteins definiert werden. Dieser Befehl ist wirkungslos, wenn ein Hauptprogramm der Abschnitte "**Anfang**" oder "**Ende**" oder ein AS-Programme im Fenster des Programm-Managers ausgewählt ist.
Unterprogramme, Funktionen oder Funktionsbausteine können bis zu **32** Parameter besitzen (Ein- oder Ausgänge). Funktionen und Unterprogramme besitzen jeweils einen einzigen Returnparameter, der den gleichen Namen wie die Funktion haben muß (gemäß der ST-Schreibkonventionen).
Die Liste oben links im Fenster zeigt die Parameter in der Reihenfolge des Aufrufmodells: zuerst die Aufrufparameter, zuletzt die Returnparameter. In der unteren Fensterhälfte erscheint die ausführliche Beschreibung des in der Liste ausgewählten Parameters. Für Parameter kommen alle ISaGRAF Datentypen in Frage. Die Returnparameter müssen am Ende der Liste hinter den Aufrufparametern erscheinen. Bei der Parameterbenennung gelten die folgenden Regeln:

- Der Name muß sich auf **16** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Die weiteren Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Der Befehl "**Hinzufügen**" wird benutzt, um einen neuen Parameter vor dem in der Liste ausgewählten Parameter einzufügen. Mit dem Befehl "**Löschen**" kann der ausgewählte Parameter gelöscht werden. Mit dem Befehl "**Sortieren**" können die Parameter automatisch sortiert werden, so daß die Returnparameter am Ende der Liste erscheinen.

Ein Programm in der Hierarchiestruktur verschieben

Der Befehl "**Umbenennen/Verschieben**" im "**Datei**"-Menü wird benutzt, um den Namen eines Programms zu ändern oder um das Programm in einen anderen Abschnitt der hierarchischen Baumstruktur zu verschieben. Die Programmiersprache eines bestehenden Programms kann jedoch nicht geändert werden. Wenn dieser Befehl eingeleitet wird, öffnet sich ein Fenster, das auch für die Programmerstellung benutzt wird. Alle Textfelder erscheinen mit den Attributen des ausgewählten Programms. Hier kann der Name des Programms geändert oder ein anderer Abschnitt oder ein anderes Vaterprogramm ausgewählt werden, um das Programm in der hierarchischen Baumstruktur zu verschieben.
Der Befehl "**Programme sortieren**" im "**Datei**"-Menü wird benutzt, um die Programme einer gleichen Ebene, welche demselben Vaterprogramm untergeordnet sind, explizit zu ordnen. Wenn sich das ausgewählte Programm in der Hauptebene befindet, ordnet der Befehl die Hauptprogramme des ausgewählten Abschnitts. Wenn sich das ausgewählte Programm in einer Unterebene befindet, ordnet der Befehl nur die AS-Sohnprogramme und Unterprogramme mit demselben Vaterprogramm wie das ausgewählte Programm. Zuerst wird das zu verschiebende Programm im Dialogfeld "**Programme sortieren**" ausgewählt. Dann klickt man auf die Schaltflächen "**Nach oben**" oder "**Nach unten**", um es in der Liste zu verschieben.

Programme kopieren

Um ein Programm zu kopieren , wählen Sie das Ursprungsprogramm aus der Programmliste und starten den Befehl "**Datei / Kopieren**". Wenn dieser Befehl eingeleitet wird, öffnet sich ein Fenster, das auch für die Programmerstellung benutzt wird. Alle Textfelder erscheinen mit den Attributen des ausgewählten Programms. Geben Sie den Namen des Zielprogramms und seinen Standort in den Abschnitten der hierarchischen Baumstruktur ein. Wenn das Zielprogramm noch nicht existiert, wird es an der spezifizierten Stelle erstellt. Wenn es das Zielprogramm schon gibt, wird es überschrieben. Das Programm wird mit allen seinen lokalen Deklarationen und Definitionen kopiert. Die Programmiersprache des Zielprogramms muß die gleiche wie die des Ursprungsprogramms sein. Klicken Sie "**OK**", um das Programm zu kopieren.
Der Befehl "**In einem anderen Projekt kopieren**" im "**Datei**"-Menü kopiert das ausgewählte Programm unter dem gleichen Namen in einem anderen Projekt. Die AS-Sohnprogramme und Unterprogramme des ausgewählten Programms können mit ihm kopiert werden. Der Name des ausgewählten Programms und die Namen seiner Sohnprogramme dürfen im Zielprojekt nicht benutzt werden. Programme können mit diesem Befehl nicht überschrieben werden. Alle lokalen Deklarationen und Definitionen werden mit den Programmen kopiert.

Programme löschen

Um ein Programm zu löschen, wählt man es in der Programmliste aus und leitet den Befehl "**Datei / Löschen**" ein. Bei einem Programm mit Sohn- oder Unterprogrammen müssen zuerst die Sohn- und Unterprogramme gelöscht werden. Alle lokalen Deklarationen und Definitionen werden mit dem Programm gelöscht.

Funktionen und Funktionsbausteine aus der Bibliothek importieren

Der Befehl "**Werkzeuge / Aus der Bibliothek importieren**" wird benutzt, um in IEC-Sprachen geschriebene Funktionen oder Funktionsbausteine aus der Bibliothek in den Abschnitten "**Funktionen**" oder "**Funktionsbausteine**" des offenen Projekts zu kopieren. Die mit den importierten Funktionen verknüpften lokalen Variablen und Definitionen werden mitkopiert. Eine aus der Bibliothek importierte Funktion kann in einen anderen Abschnitt oder an eine andere Stelle in der hierarchischen Baumstruktur verschoben werden (Befehl "**Datei / Umbenennen/Verschieben**"). Um Namenskollisionen zu vermeiden, müssen importierte Funktionen oder Funktionsbausteine umbenannt werden, wenn sie in das Projektfeld importiert werden. Achtung: Bei einer Funktion muß auch der Returnparameter umbenannt werden.

Funktionen und Funktionsbausteine in die Bibliothek exportieren

Der Befehl "**Werkzeuge / In die Bibliothek exportieren**" wird benutzt, um ein Programm der Abschnitte "**Funktionen**" oder "**Funktionsbausteine**" (des offenen Projekts) in die entsprechende Bibliothek zu exportieren. Die mit den exportierten Funktionen oder Funktionsbausteinen verknüpften lokalen Variablen und Definitionen werden mitkopiert. Exportierte Funktionen oder Funktionsbausteine müssen vom ISaGRAF Bibliotheksmanager neu kompiliert werden, um zu prüfen, ob sie in der Bibliotheksumgebung benutzt werden können. Funktionen und Funktionsbausteine der Bibliothek können keine globalen Variablen benutzen.

A.3.3. Werkzeuge für die Code-Entwicklung

Die Befehle des Menüs "**Codierung**" werden benutzt, um den Codegenerator zu starten und um Optionen und zusätzliche Daten für die Entwicklung des Anwendungscodes einzugeben. Siehe Kapitel "**Codegenerator benutzen**" in diesem Dokument für weitere Informationen zu diesen Werkzeugen.

Anwendungscode entwickeln

Der Befehl "**Codierung**" startet die Code-Entwicklung des Projekts. Bevor man diesen Befehl benutzen kann, müssen die Optionen für die Entwicklung des Zielcodes korrekt eingestellt werden. Vor der Entwicklung des Zielcodes werden alle bis dahin noch nicht geprüften Programme nach Syntaxfehlern überprüft. ISaGRAF enthält einen Inkremantal-Compiler, der bereits kompilierte Programme nicht erneut kompiliert.

Das ausgewählte Programm prüfen

Mit dem Befehl "**Prüfen**" kann der Benutzer die Programmsyntax des in der Liste ausgewählten Programms überprüfen. Wenn nach beendeter Prüfung keine Fehler erkannt wurden, wird das Programm nicht erneut bei der Code-Entwicklung geprüft, außer sein Inhalt oder die mit ihm verknüpften Variablen oder Definitionen haben sich geändert.

Eine Änderung simulieren

Der Befehl "**Änderung simulieren**" simuliert eine Änderung für jedes Programm, so daß sämtliche Programme bei der nächsten Code-Entwicklung erneut kompiliert werden.

Ausführungsoptionen

Dieser Befehl öffnet ein Dialogfeld, in dem die hauptsächlichen Ausführungsparameter einer Anwendung eingegeben werden, einschließlich Zykluszeit-Programmierung, Verwaltung der Ausführungsfehler, Startmodus und Hardware-Implementierung der nicht-flüchtigen Variablen. Siehe Kapitel "Codegenerator benutzen" in diesem Dokument für weitere Informationen zu diesem Befehl.

Compileroptionen

Dieser Befehl wird benutzt, um die Optionen des ISaGRAF Codegenerators für die Entwicklung und Optimierung des Zielcodes einzustellen. Siehe Kapitel "Codegenerator benutzen" in diesem Dokument für weitere Informationen zu diesem Befehl.

Resourcen definieren

"**Resourcen**" sind benutzerdefinierte Daten (beispielsweise eine Datei), die mit dem Zielcode verbunden werden müssen, so daß sie mit ihm geladen werden können. Siehe Kapitel "Codegenerator benutzen" in diesem Dokument für weitere Informationen zum Format der Resourcen-Definitionsdatei.

A.3.4. Andere ISaGRAF Werkzeuge

Mit den Befehlen im Menü "**Projekt**" können ISaGRAF Werkzeuge für das ausgewählte Projekt gestartet werden. Siehe auch die betreffenden Kapitel dieses Dokuments für weitere Informationen zu diesen Werkzeugen.

E/A-Variablen verdrahten

Der Befehl "**E/A-Verdrahtung**" startet den ISaGRAF Editor für die Verdrahtung der E/A-Variablen. Mit diesem Werkzeug werden die im Datenverzeichnis des Projekts deklarierten E/A-Variablen mit der entsprechenden E/A-Hardware verdrahtet.

Crossreferenzen-Editor benutzen

Mit dem Befehl "**Crossreferenzen**" kann der Benutzer die Crossreferenzen des Projekts berechnen, visualisieren oder ausdrucken. Die Crossreferenzen veranschaulichen sämtliche Vorkommen einer jeden Variablen in den Quellcodes der Programme für das gesamte Projekt. Diese Funktion ist sehr nützlich, wenn man auf eine Variable oder eine beliebige globale Resource zugreifen will oder wenn sämtliche Quellcode-Vorkommen einer globalen Variablen aufgelistet werden sollen.

Eingaben im Projektkopf

Mit dem Befehl "**Projektkopf**" wird die Bearbeitung des Projektkopfes eingeleitet. Dieses Dokument dient der genauen Beschreibung eines Projekts und unterscheidet es von den anderen Projekten in der Liste. Im Projektkopf können außerdem laufende Angaben zur Projektentwicklung erfaßt werden. Der Projektkopf wird im Fenster des Projektmanagers angezeigt.

Eine komplette Dokumentation ausdrucken

Der Befehl "**Projektdokumentation drucken**" dient dem Aufbau und Ausdruck einer kompletten Dokumentation für das ausgewählte Projekt, in der alle Projektkomponenten (Programme, Variablen, Parameter...) zusammengefaßt werden. Wenn eine spezielle (unvollständige) Dokumentation gewünscht wird, kann ein Inhaltsverzeichnis definiert werden.

Projektprotokoll

Dieser Befehl öffnet ein Dialogfeld, in dem das Projektprotokoll angezeigt wird. Siehe Kapitel "Projektverwaltung" in diesem Dokument für weitere Informationen zu diesem Befehl.

A.3.5. Dem Werkzeugmenü Befehle hinzufügen

ISaGRAF ermöglicht, andere Befehle in das "**Werkzeug**"-Menü einzufügen, z.B. benutzerdefinierte Befehle, die in der Textdatei "**\ISAWIN\COM\ISA.MNU**" beschrieben werden. Es können bis zu 10 Befehle hinzugefügt werden. Kommentare können auf jeder Zeile eingefügt werden. Sie werden mit dem Zeichen "**;**" eingeleitet. Jeder Kommentar wird auf zwei Textzeilen geschrieben, gemäß der folgenden Syntax:

```
M=menu_string
C=command_line
```

Menu_string (Menü-Zeichenkette) ist der Text, der im "**Werkzeug**"-Menü angezeigt wird. Command_line (Befehlszeile) ist eine beliebige ausführbare MS-DOS- oder Windows-Datei und kann mit Argumenten vervollständigt werden. In der Befehlszeile kann das Zeichen "**%A**" benutzt werden, um den Namen des offenen Projekts zu ersetzen, und das Zeichen "**%P**", um den Namen des ausgewählten Programms zu ersetzen. Das folgende Beispiel startet "Notepad", um das ausgewählte Programm zu bearbeiten (für ST- und AWL-Programme):

```
M=Mit Notepad bearbeiten
C=Notepad.exe \isawin\apl\%A\%P.lsf
```

A.3.6. Anwendungen simulieren und debuggen

Die Befehle des Menüs "**Debugging**" werden benutzt, um den grafischen ISaGRAF Debugger zu starten, entweder im Simulationsmodus oder im Onlinemodus.

Simulation

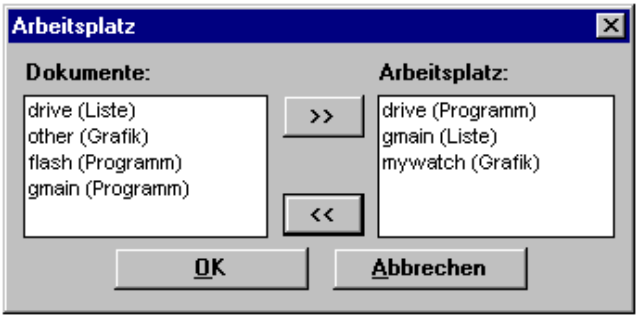
Der Befehl "**Simulieren**" öffnet den Debugger im Simulationsmodus. In diesem Modus erscheint ein anderes Fenster, das Simulator genannt wird. Dieser Befehl wird benutzt, um eine Anwendung zu testen, wenn noch keine Zielsteuerung verfügbar ist. Wenn der Simulator gestartet wird, schließt sich das Fenster des Programm-Managers. Es öffnet sich dann wieder im Debugging-Modus, wenn die Debugger- und Simulator-Fenster offen sind. Der Simulator kann nicht starten, wenn noch kein Zielcode generiert ist. Der Simulator kann auch dann nicht starten, wenn noch untergeordnete Fenster (Editoren, Code-Entwicklung, E/A-Verdrahtung...) offen sind. Schließen Sie diese Fenster, bevor Sie den Befehl einleiten. Dieser Befehl ist auch in den Menüs der ISaGRAF Editoren verfügbar.

Online-Debugging

Der Befehl "**Debuggen**" öffnet das Fenster des Debuggers und schließt das Fenster des Programm-Managers. Es öffnet sich dann wieder im Debugging-Modus, wenn die Kommunikation zwischen dem Debugger und der Zielanwendung hergestellt ist. Der Debugger kann nicht starten, wenn noch kein Zielcode generiert ist. Der Debugger kann auch dann nicht starten, wenn noch untergeordnete Fenster (Editoren, Code-Entwicklung, E/A-Verdrahtung...) offen sind. Schließen Sie diese Fenster, bevor Sie den Befehl einleiten. Dieser Befehl ist auch in den Menüs der ISaGRAF Editoren verfügbar.

Vorbereitung des Debugging-Arbeitsbereichs

Mit dem Befehl "**Debugging / Arbeitsbereich**" können verschiedene Dokumente für den Arbeitsbereich definiert werden, beispielsweise Programme, SpotLight Grafiken und Variablenlisten. Grafiken und Steuerdiagrammlisten früherer ISaGRAF Versionen werden ebenfalls in den Projektdokumenten aufgelistet. Die für den Arbeitsbereich definierten Dokumente werden automatisch geöffnet, wenn die Simulation oder Online-Überwachung eingeleitet wird.



Links im Dialogfeld erscheinen die bestehenden Projektdokumente, rechts die für den Arbeitsbreich ausgewählten Dokumente. Benutzen Sie die Schaltflächen ">>" und "<<", um Dokumente von einer Liste zur anderen zu übertragen. Jedes Projekt besitzt seine eigene Dokumentliste für den Arbeitsbereich.

Kommunikationsverbindung

Der Befehl "**Kommunikationsparameter**" ermöglicht dem Benutzer, die Parameter für die Kommunikation zwischen dem Debugger des Hostrechners und dem ISaGRAF Zielsystem einzugeben.
Die "**Slave-Adresse**" identifiziert das ISaGRAF Zielsystem oder den Task. Diese Nummer liegt zwischen **1** und **255**. Siehe Herstellerhandbuch für die Slave-Adresse des jeweiligen Zielsystems.
Das Feld "**Anschluß**" identifiziert die Hardware zwischen der ISaGRAF Workstation und der Zielsteuerung. Dies kann entweder der Name eines seriellen Anschlusses oder "**Ethernet**" sein, eine reservierte TCP-IP Kommunikation, die "Winsock" Version 1.1 benutzt.
Das "**Time-Out**" ist die Zeitspanne, die dem Zielsystem für die Kommunikationsoperationen gewährt wird, und zwar vom Ende einer Anfrage des Debuggers bis zum Anfang der Antwort des Zielsystems. Diese Zeitspanne wird in **Millisekunden** ausgedrückt. Das Feld "Versuche" bestimmt die Anzahl der automatischen Kommunikationsversuche, die der Debugger ausführt, bevor ein Kommunikationsfehler erkannt wird.

Serieller Anschluß

Nach Auswahl eines seriellen Anschlusses (COM1..4) wird die Schaltfläche "**Konfigurieren**" benutzt, um auf die Kommunikationsparameter des seriellen Anschlusses zuzugreifen.
Übertragungsgeschwindigkeit, Parität und Format können ausgewählt werden. Wenn "**Hardware**" für den "**Datenfluß**" gewählt wird, steuert die ISaGRAF Workstation die CTS- und DSR-Signale, um Hardware-Handshaking während des Datenaustausches zu ermöglichen.

Ethernet-Anschluß

Wenn "Ethernet" als Kommunikationsschnittstelle gewählt wird, wird die Schaltfläche "**Konfigurieren**" dazu benutzt, die "Internet-Adresse" und die Nummer der "Internet-Schnittstelle" für die TCP-IP-Kommunikation einzugeben.
Diese Felder benutzen Standardformate, die von der Socket-Schnittstelle definiert werden. Die Workstation benutzt die WINSOCK.DLL Version 1.1 Bibliothek für TCP-IP-Kommunikationen. Diese Datei muß ordungsgemäß auf der Festplatte installiert werden. Falls nicht spezifiziert, wird "**1100**" als Nummer für die benutzte Schnittstelle beim Start der ISaGRAF Zielsteuerung vorgegeben.

A.4. AS-Editor benutzen

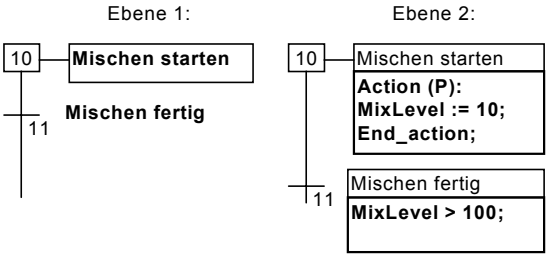
Die AS-Sprache beschreibt die Operationen eines sequentiellen Prozesses. Sie ist eine einfache grafische Darstellung der einzelnen Prozeßschritte sowie der Bedingungen, mit denen die aktiven Schritte verändert werden können. AS-Programme werden mit Hilfe des ISaGRAF AS-Grafikeditors eingegeben. Die AS ist die Kernsprache der IEC 1131-3 Norm. Die anderen Sprachen beschreiben gewöhnlich die Aktionen in den Schritten und die logischen Bedingungen für die Transitionen. Der ISaGRAF AS-Editor kombiniert Grafik- und Textbearbeitung, so daß komplette AS-Programme mit ihren grafischen AS-Netzwerken und der Beschreibung der entsprechenden Aktionen und Bedingungen in einem Arbeitsgang erfaßt werden können.

A.4.1. Eigenschaften der AS-Sprache

Die AS-Sprache dient der Darstellung sequentieller Prozesse. Sie zerlegt den Prozeßzyklus in eine Anzahl genau definierter, aufeinanderfolgender **Schritte** (stabile Situationen), die durch **Transitionen** getrennt sind. Siehe ISaGRAF Sprachreferenzen für weitere Informationen zur AS-Sprache.
Im AS-Netzwerk werden die einzelnen Komponenten durch **gerichtete Verbindungslinien** miteinander verbunden, die in einer vorgegebenen Richtung von **oben** nach **unten** verlaufen. Es folgen die hauptsächlichen grafischen Komponenten zum Aufbau eines AS-Netzwerks:

 ..... Initialisierungsschritt
 ..... Schritt
 ..... Transition
 ..... Sprungbefehl zu einem Schritt
 ..... Makroschritt
 ..... Anfangsschritt eines Makroschritts
 ..... Endschritt eines Makroschritts

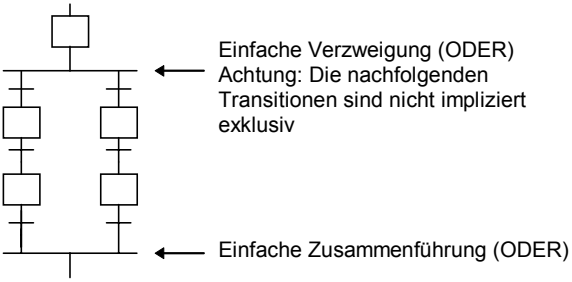
Die AS-Programmierung erfolgt gewöhnlich auf zwei Ebenen: Die **Ebene 1** zeigt das AS-Netzwerk, die Referenznummern der Schritte und Transitionen, sowie die mit den Schritten und Transitionen verbundenen Kommentare. Die **Ebene 2** ist die **ST**- oder **AWL**-Programmierung der in den Schritten enthaltenen Aktionen oder der mit den Transitionen verbundenen Bedingungen. Diese Aktionen und Bedingungen können in anderen Programmiersprachen (**FBS**, **KOP**, **ST** oder **AWL**) geschriebene **Unterprogramme** aufrufen. Das folgende Beispiel veranschaulicht die Programmierung der Ebenen 1 und 2:



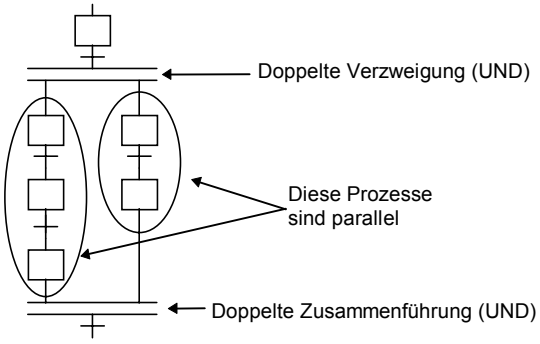
Die Programmierung der Ebene 2 eines Schritts erfolgt mit einem Texteditor. Sie kann in ST oder AWL geschriebene Aktionsblöcke einschließen. Die Programmierung der Ebene 2 einer Transition erfolgt entweder in den Textsprachen AWL oder ST oder mit dem Schnellen KOP-Editor.

Verzweigungen und Zusammenführungen

Verzweigungen und Zusammenführungen werden benutzt, um **Mehrfachverbindungen** zwischen den Schritten und Transitionen darzustellen. Einfache Verzweigungen und Zusammenführungen stellen unterschiedliche **inklusive** Möglichkeiten zwischen den verschiedenen untergeordneten Abschnitten eines Prozesses dar.

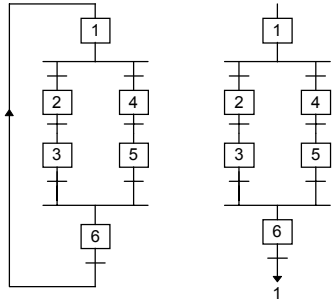


Doppelte Verzweigungen stellen **parallele** Prozesse dar.



Sprung zu einem Schritt

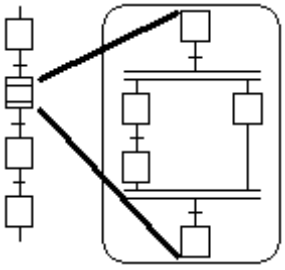
Mit dem AS-Editor können lediglich Verbindungslinien von **oben** nach **unten** gezogen werden. Eine Verbindung zu einem oberhalb im Netzwerk befindlichen Abschnitt kann mit einem **Sprungbefehl** zu einem Schritt dargestellt werden. Die folgenden Netzwerke sind äquivalent:



Sprungbefehle zu Transitionen sind unzulässig und werden expliziert als doppelte Zusammenführungen (UND) dargestellt.

Makroschritte

Makroschritte sind **einmalige** Darstellungen **unabhängiger** Gruppen von Schritten und Transitionen. Makroschritte beginnen jeweils mit einem **Anfangsschritt** und enden mit einem **Endschritt**.



Makroschritte müssen an einer anderen Stelle in ihrem AS-Programm ausführlich beschrieben werden. Das Symbol eines Makroschritts und sein Anfangsschritt müssen die gleichen **Referenznummern** besitzen. Die Beschreibung eines Makroschritts kann weitere Makroschritte beinhalten.

A.4.2. Eingabe eines AS-Netzwerks

Der Aufbau eines AS-Netzwerks erfolgt ganz einfach durch Einsetzen der einzelnen Komponenten. Alle einfachen Verbindungslinien (horizontal oder vertikal) für die Verbindung von zwei Elementen werden automatisch vom AS-Editor hinzugefügt. Um ein AS-Element in das Netzwerk einzusetzen, verlagert man den Eingabecursor

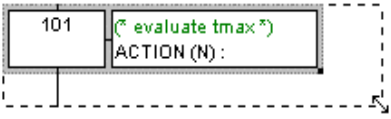
an die gewünschte Stelle im Eingaberaster und wählt den entsprechenden Typ in der Editor-Toolbox. Das Symbol erscheint an der aktuellen Position. Es können auch die folgenden Tastenkombinationen benutzt werden:

F2:	Einsetzen eines Initialisierungsschritts
F3:	Einsetzen eines einzelnen Schritts
F4:	Einsetzen einer Transition
F5:	Einsetzen eines Sprungbefehls zu einem Schritt
F6: F7:	Einsetzen einer ODER Verzweigung oder Zusammenführung / Hinzufügen von Zweigen
↑F6: ↑F7:	Einsetzen einer UND Verzweigung oder Zusammenführung / Hinzufügen von Zweigen
F8:	Einsetzen eines Makroschritts
F9: ↑F9:	Einsetzen eines Anfangs- oder Endschritts für den Rumpf eines Makroschritts

(Das Symbol "↑" steht für eine Kombination mit der SHIFT-Taste)

Das **Bearbeitungsraster** besteht aus **Matrixzellen**. Der AS-Editor verfügt über eine Schaltoption, mit der das Raster bei der Eingabe eines Netzwerks entweder angezeigt oder versteckt werden kann. Arbeiten im Raster ist beim Layout eines Netzwerks vorteilhaft oder wenn untergeordnete Abschnitte eines Netzwerks ausgewählt werden sollen. Benutzen Sie den Befehl "**Optionen / Layout**", um das Raster anzuzeigen oder zu verstecken.

Die aktuelle Position ist jederzeit in der Matrix des ISaGRAF AS-Editors ersichtlich. Die aktive Zelle erscheint grau markiert und verfügt in der rechten unteren Ecke über ein kleines Kästchen, mit dem die Zellen auf die gewünschte Größe gebracht werden können. Auch der X/Y-Quotient der Zellen kann auf diese Weise modifiziert werden.



F6: ↑F6: *Eine Verzweigung oder Zusammenführung erstellen*

Verzweigungen und Zusammenführungen werden immer **von links nach rechts** gezeichnet. Um eine Verzweigung oder Zusammenführung zu zeichnen, wird ihr **linker Zweig** in das Netzwerkfeld eingesetzt. Der Typ (einfach oder doppelt) wird durch Auswahl einer dieser Schaltflächen in der Toolbox bestimmt.

F6: ↑F6: F7: ↑F7: *Zweige hinzufügen*

Die **Anfangs-** und **Endposition** eines jeden **Nebenzweigs** werden mit Hilfe dieser Toolbox-Schaltflächen auf die Verzweigungs- oder Zusammenführungslinie gesetzt. Die linke Ecke einer Verzweigung oder Zusammenführung muß bereits vorhanden sein, bevor neue Zweige hinzugefügt werden können. Die rechten Ecken erscheinen

im gleichen Stil (einfach oder doppelt) wie die linke Hauptecke. Rechte Ecken können erst hinzugefügt werden, wenn die linke Hauptecke bereits existiert.

Einen Makroschritt einfügen

Diese Schaltfläche wird benutzt, um einen Makroschritt in ein Hauptdiagramm einzufügen. Der Rumpf des Makroschritts muß an einer anderen Stelle im betreffenden AS-Programm eingegeben werden.

Rumpf des Makroschritts

Makroschritte werden in dem AS-Programm beschrieben, in dem sich auch das Hauptdiagramm befindet. Makroschritte beginnen jeweils mit einem **Anfangsschritt** und enden mit einem **Endschritt**. Das untergeordnete Diagramm für die Makroimplementierung muß vollkommen **unabhängig** sein. Der Anfangsschritt eines Makroschritts muß die gleiche **Referenznummer** wie das Symbol dieses Makroschritts im Hauptdiagramm besitzen.

A.4.3. Ein bestehendes AS-Netzwerk bearbeiten

Die Auswahl eines rechteckigen Felds im Netzwerk erfolgt mit der Maus oder mit den Pfeiltasten der Tastatur. Das ausgewählte Feld erscheint grau getönt. Jetzt können folgende Befehle im Menü "**Bearbeiten**" benutzt werden:

Ausschneiden / Kopieren / Löschen / Einfügen

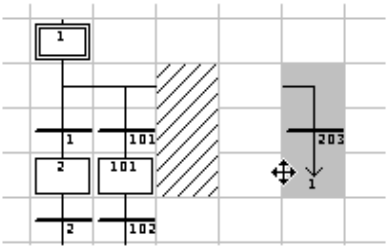
Wenn die **"Pfeil"**-Schaltfläche in der Editor-Toolbox aktiv ist, sind folgende Befehle im Menü **"Bearbeiten"** verfügbar:

Ausschneiden..... Verlagert das ausgewählte Feld vom Bildschirm in die AS-Zwischenablage
Kopieren..... Kopiert das ausgewählte Feld vom Bildschirm in die AS-Zwischenablage
Löschen ..... Löscht das ausgewählte Feld
Einfügen..... Fügt den Inhalt der AS-Zwischenablage an der aktuellen Position ein

Der Befehl "**Bearbeiten / Einfügen**" kopiert die AS-Zwischenablage auf dem Bildschirm. Die Befehle Kopieren / Einfügen können sowohl im AS-Netzwerk, als auch bei der Programmierung der Ebene 2 eines Schritts oder einer Transition benutzt werden. Es ist auch möglich, ein Netzwerk eines Programms zu kopieren und dann in ein anderes AS-Programm einzufügen. Die Elemente werden vor der aktuellen Position des Eingabecursors eingefügt.

Elemente verschieben

Wenn AS-Elemente in einem AS-Netzwerk ausgewählt sind, können sie im Netzwerk verschoben werden, indem man das ausgewählte Feld mit der Maus an eine andere Stelle zieht. Während das Auswahlfeld gezogen wird, erscheint die Ursprungsposition der ausgewählten Elemente schraffiert.



Die Zielposition der ausgewählten Elemente muß leer sein. Beim Verschieben von AS-Elementen ist kein Einfügen möglich.

Schritte und Transitionen neu nummerieren

Sämtliche Schritte und Transitionen eines AS-Netzwerks besitzen logische Identifizierungsnummern. Der Befehl "**Bearbeiten / Neu nummerieren**" versieht alle Schritte und Transitionen des in Arbeit befindlichen AS-Netzwerks automatisch mit laufenden Referenznummern. Wenn eine Schrittnummer verändert wird, werden sämtliche Sprungbefehle zu diesem Schritt automatisch mit der neuen Referenznummern aktualisiert. (Dies gilt auch für Makro- und Anfangsschritte.)

Direkter Zugriff auf Schritte und Transitionen

Der Befehl "**Bearbeiten / Gehen zu**" wird benutzt, um auf bestehende Schritte und Transitionen zuzugreifen. Die Bildlaufleiste wird automatisch so ausgerichtet, daß der Schritt oder die Transition auf dem Bildschirm sichtbar ist.

Text suchen und ersetzen

Der Befehl "**Bearbeiten / Suchen Ersetzen**" wird benutzt, um Zeichenketten im gesamten Programm (in allen Schritten und Transitionen) zu suchen und zu ersetzen. Der im Dialogfeld "Suchen/Ersetzen" eingegebene Text wird in der Programmierebene 2 gesucht. Die Ebene 2 öffnet sich automatisch.

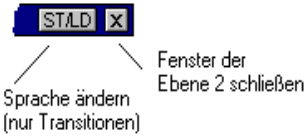
A.4.4. Die Programmierung der Ebene 2

Die Texteingabe der Ebene 2 erfolgt durch Doppelklicken auf ein Schritt- oder Transitionssymbol. Das Programmierfeld der Ebene 2 erscheint rechts im AS-Fenster. Die Trennlinie zwischen AS-Netzwerk und Ebene 2 kann beliebig verschoben werden.
Es besteht die Möglichkeit, ein oder zwei Programmierfelder der Ebene 2 gleichzeitig zu öffnen. Die folgenden Befehle sind auf der Tastatur, mit der Maus oder im Menü "Bearbeiten" verfügbar:

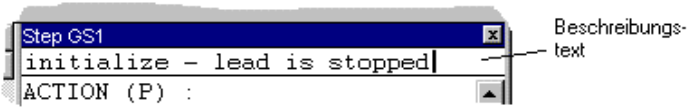
Tastatur	*Maus*	*Menü "Bearbeiten"*
Eingabe	Doppelklicken	Ebene 2 bearbeiten
Strg+Eingabe	Strg + Doppelklicken	Ebene 2 in separatem Fenster bearbeiten

Wenn zwei Fenster der Ebene 2 sichtbar sind, kann die Trennline zwischen den beiden Fenstern beliebig verschoben werden. Die Schaltfläche rechts in der Titelzeile schließt das Fenster der Ebene 2.

Für die Programmierung der Ebene 2 wird **ST** (Strukturierter Text) vorgegeben. Die Ebene 2 einer Transition kann aber auch mit dem **Schnellen KOP**-Editor programmiert werden. Die betreffende Sprache wird mit Hilfe der Schaltfläche "**ST/KOP**" in der Titelzeile der Ebene 2 aktiviert. Dieser Befehl ist nur verfügbar, wenn das Fenster der Ebene 2 leer ist.



Ein einzeiliges Eingabefeld erscheint oberhalb des Fensters der Ebene 2. Es dient der Eingabe einer kurzen Beschreibung. Dieser Text wird als IEC-Kommentar in Diagrammen von AS-Symbolen angezeigt. Er ist sehr nützlich, da er von bestimmten Befehlen wie zum Beispiel "Gehen zu..." benutzt wird und in AS-Ausdrücken die Schritte und Transitionen zu dokumentiert.



Der Befehl "**Optionen / Aktualisieren**" kann jederzeit benutzt werden, wenn das Fenster der Ebene 2 offen ist, um das AS-Netzwerk mit den Modifizierungen der Ebene 2 zu aktualisieren.

Einen Variablennamen einfügen

Wenn Sie in einer Textsprache programmieren, können Sie durch Klicken auf diese Schaltfläche eine im Datenverzeichnis des Projekts deklarierte Variable auswählen und ihren Namen am Standort des Eingabecursors einfügen. Wenn Sie im Schnellen KOP programmieren, können Sie durch Klicken auf diese Schaltfläche eine Variable auswählen, die dann mit dem ausgewählten Kontakt oder E/A-Parameter eines Funktionsbausteins verbunden wird.

Eine Impulsaktion in einen Schritt einfügen

Bei der Programmierung der Ebene 2 eines Schritts können Sie durch Klicken dieser Schaltfläche das Modell einer Impulsaktion am aktuellen Standort des Eingabecursors einfügen. Es folgt das Format einer Impulsaktion:

```
Action (P) :
   ST statement;
   ...
End_Action;
```

Impulsaktionen sind Anweisungen, die nur einmal ausgeführt werden, wenn der Schritt aktiv wird. Siehe ISaGRAF Sprachreferenzen für weitere Informationen zur AS-Programmierung.

Eine nicht-gespeicherte Aktion in einen Schritt einfügen

Bei der Programmierung der Ebene 2 eines Schritts können Sie durch Klicken dieser Schaltfläche das Modell einer nicht-gespeicherten Aktion am aktuellen Standort des Eingabecursors einfügen. Es folgt das Format einer nicht-gespeicherten Aktion:

```
Action (N) :
   ST statement;
   ...
End_Action;
```

Nicht-gespeicherte Aktionen sind Anweisungen, die in jedem SPS-Zyklus ausgeführt werden, wenn der Schritt aktiv ist. Siehe ISaGRAF Sprachreferenzen für weitere Informationen zur AS-Programmierung.

Neue P0 und P1 Aktionsqualifizierer

ISaGRAF unterstützt neue **P0-** und **P1**-Aktionsqualifizierer. Bei der Programmierung der Ebene 2 eines Schritts können Sie durch Klicken der entsprechenden Schaltfläche das Modell eines P0- oder P1-Aktionsblocks am aktuellen Standort des Eingabecursors einfügen. Es folgt das Format solcher Blöcke:

```
Action (P0) :              Action (P1) :
   ST statement;                  ST statement;
   ...                            ...
End_Action;                End_Action;
```

P1-Aktionen sind Anweisungen, die nur einmal ausgeführt werden, wenn der Schritt aktiv wird (wie bei den Impulsaktionen). P0-Aktionen sind Anweisungen, die nur einmal ausgeführt werden, wenn der Schritt passiv wird. Siehe ISaGRAF Sprachreferenzen für weitere Informationen zur AS-Programmierung.

Boolesche Aktionen

Eine andere Semantik ist verfügbar, um unmittelbar auf eine boolesche Variable, entsprechend der Aktivität des Schritts, einzuwirken. Diese Aktionen weisen einer booleschen Variablen vom Typ intern oder Ausgang ein **Schrittaktivitätssignal** zu. Es folgt die Syntax boolescher Aktionen:

<boolesche_variable> (N);	weist der Variablen das Schrittaktivitätssignal zu
<boolesche_variable>;	gleiche Wirkung (das Attribut N ist optionell)
/ <boolesche_variable>;	weist der Variablen die Negation des Schrittaktivitätssignals zu

Weitere Funktionalitäten sind verfügbar, um eine boolesche Variablen zu setzen oder rückzusetzen, wenn der Schritt aktiv wird. Hier die Syntax der booleschen Setze- und Rücksetzeaktionen:

<boolesche_variable> (S);	setzt die Variable auf TRUE, wenn das Schrittaktivitätssignal TRUE wird
<boolesche_variable> (R);	setzt die Variable auf FALSE, wenn das Schrittaktivitätssignal TRUE wird

AS-Aktionen

Eine andere Semantik ist verfügbar, um die Ausführung eines AS-Sohnprogramms zu steuern. Eine AS-Aktion ist eine AS-Sohnsequenz, die entsprechend der Bedingung des Schrittaktivitätssignals gestartet oder gestoppt wird. AS-Aktionen können die Qualifizierer **N** (nicht-gespeichert), **S** (setzen), oder **R** (rücksetzen) besitzen. Es folgt die Syntax von AS-Aktionen:

<sohn_programm> (N);	startet die Sohnsequenz, wenn der Schritt aktiv wird und stoppt die Sohnsequenz, wenn der Schritt passiv wird
< sohn_programm>;	gleiche Wirkung wie vorangehende Syntax (Attribut N ist optionell)
< sohn_programm> (S);	startet die Sohnsequenz, wenn der Schritt aktiv wird - nichts geschieht, wenn der Schritt passiv wird
< sohn_programm> (R);	stoppt die Sohnsequenz, wenn der Schritt aktiv wird - nichts geschieht, wenn der Schritt passiv wird

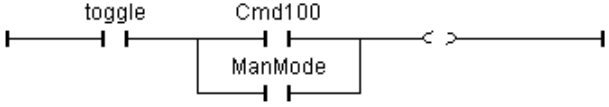
Die als Aktion spezifizierte AS-Sequenz muß ein bestehendes **AS-Sohnprogramm** des in Arbeit befindlichen, mit dem ISaGRAF Programmanager erstellten Programms sein.

In ST geschriebene Transitionen

Die Ebene 2 einer Transition ist ein boolescher Ausdruck. Um diesen Ausdruck in der ST-Sprache zu programmieren, wird die boolesche Bedingung einfach in der ST-Syntax eingegeben. Wahlweise kann ein Semikolon am Ende des Ausdrucks hinzugefügt werden.

Im schnellen KOP geschriebene Transitionen

Die Bedingung der Ebene 2 einer Transition kann auch mit dem Schnellen KOP-Editor programmiert werden. In diesem Fall besteht der Kontaktplan aus nur einem Rung, mit einer einzigen Spule, welche die Transition darstellt. Der Name der Transition wird nicht mit dem Spulensymbol wiederholt. Hier das Beispiel einer Transitionsbedingung, die im Schnellen KOP programmiert wurde.



Beim Programmieren im Schnellen KOP wird der Eingabecursor im logischen Programmierraster mit Hilfe der Pfeiltasten verlagert. Dann werden die Symbole eingesetzt, und zwar durch Anschlagen der folgenden Funktionstasten:

F2:..... fügt einen Kontakt hinter dem ausgewählten Symbol ein / initiiert das Rung
F3:..... fügt einen Kontakt vor dem ausgewählten Symbol ein
F4:..... fügt einen Kontakt parallel zum ausgewählten Symbol ein
F6:..... fügt einen Baustein hinter dem ausgewählten Symbol ein
F7:..... fügt einen Baustein vor dem ausgewählten Symbol ein

F8:..... fügt einen Baustein parallel zum ausgewählten Symbol ein
Anstatt die Funktionstasten der Tastatur zu benutzen, können Sie auch auf die Funktionstastenleiste im Fenster der Ebene 2 klicken.

Drücken Sie die RÜCKTASTE, wenn sich der Eingabecursor auf einem Kontakt oder dem E/A-Parameter eines Bausteins befindet, um eine Variable auszuwählen oder einen konstanten Wert einzugeben. Drücken Sie die RÜCKTASTE, wenn sich der Eingabecursor auf einem Funktionsbaustein befindet, um den Bausteintyp auszuwählen. Doppelklicken auf sein Symbol bewirkt das gleiche.
Drücken Sie Strg + LEERTASTE, wenn ein Kontakt ausgewählt ist, um den Kontakt- oder Spulentyp (direkt oder invertiert) zu verändern. Siehe Kapitel "Schnellen KOP-Editor benutzen" in diesem Handbuch für weitere Informationen über die Kapazitäten des Schnellen KOP.

A.4.5. AS-Album benutzen

Der AS-Editor von ISaGRAF verwaltet ein AS-Album. Dies ist eine Sammlung von AS-Strukturen, die in ein beliebiges AS-Netzwerk eingefügt werden können. Die Elemente des AS-Albums können wahlweise die Programmierung der Ebene 2 der Schritte und Transitionen miteinbeziehen. Benutzen Sie die folgenden Befehle des "**Werkzeug**"-Menüs:

Im AS-Album kopieren	kopiert die ausgewählten Elemente im AS-Album
Aus AS-Album einfügen	fügt ein Element aus dem AS-Album an der aktuellen Position ein

Beim Kopieren eines ausgewählten AS-Symbols im AS-Album (d.h. bei der Erstellung eines neuen Elements im AS-Album) kann die Programmierung der Ebene 2 wahlweise miteingebettet werden.

A.5. Flußdiagramm-Editor benutzen

Der ISaGRAF Flußdiagramm-Grafikeditor ermöglicht die Eingabe von kompletten FD-Programmen (Flußdiagrammen), mit Aktionen und Tests (Entscheidungen), die wahlweise in ST, AWL oder im Schnellen KOP programmiert werden.

A.5.1. Eigenschaften der FD-Sprache

Das **Flußdiagramm** (**FD**) ist eine grafische Sprache für die Beschreibung **sequentieller Operationen**. Flußdiagramme bestehen aus **Aktionen** und **Tests**. Der Datenfluß zwischen den Aktionen und Tests wird durch **gerichtete Verbindungslinien** dargestellt. Es folgen die grafischen Komponenten der Flußdiagramm-Sprache:

Anfang eines Flußdiagramms: Am Anfang eines FD-Programms steht ein "Begin"-Symbol. Dieses Symbol ist einmalig und darf nicht weggelassen werden. Es stellt bei seiner Aktivierung den Initialzustand des Flußdiagramms dar.

Ende eines Flußdiagramms: Das "End"-Symbol bildet den Abschluß eines Flußdiagramms. Es ist einmalig und darf nicht weggelassen werden. Dieses Symbol stellt den abschließenden Zustand des Flußdiagramms dar und muß an sein Ende gesetzt werden, selbst wenn das Diagramm in einer unendlichen Schleife ausgeführt wird und kein eigentlicher Anschluß zum "End"-Symbol besteht.

FD-Datenflußverbindungen: Die sogenannte Datenflußverbindung ist eine Linie, die den Datenfluß zwischen zwei Diagrammpunkten darstellt. Sie endet in einem Pfeil. Es ist nicht möglich, zwei Verbindungen am gleichen Quellenpunkt anzuschließen.

FD-Aktionen: Die Aktionssymbole stellen auszuführende Aktionen dar. Aktionen werden jeweils mit einer Nummer und einem Namen gekennzeichnet. Zwei unterschiedliche Objekte eines Flußdiagramms können nicht den gleichen Namen oder die gleiche logische Nummer besitzen. Die Programmiersprache für Aktionen ist wahlweise ST, KOP oder AWL. Eine Aktion besitzt stets zwei Anschlußverbindungen, davon führt eine zur Aktion hin und eine andere von der Aktion weg.

FD-Tests: Ein FD-Test stellt eine boolesche Bedingung dar und wird mit einer Nummer und einem Namen gekennzeichnet. Entsprechend der Bewertung des zugeordneten ST-, KOP- oder AWL-Ausdrucks wird der Datenfluß zu einem "YES" oder "NO" Pfad geleitet. Beim Programmieren im ST-Text kann wahlweise ein Semikolon auf den Ausdruck folgen. Beim Programmieren im KOP stellt die einzige Spule den Wert der Bedingung dar.

FD-Unterprogramme: Das System ermöglicht die Beschreibung einer Hierarchiestruktur. Die FD-Programme werden in einer hierarchischen Baumstruktur

angeordnet. Jedes FD-Programm kann andere FD-Programme aufrufen. Diese Programme bezeichnet man als Sohnprogramme des aufrufenden FD-Programms. FD-Programme, die andere FD-Programme aufrufen, werden Vaterprogramme genannt. Die FD-Programme werden in einer hierarchischen Baumstruktur miteinander verknüpft und stehen in einer "Vater-Sohn"-Beziehung zueinander. Ein Unterprogrammsymbol in einem Flußdiagramm steht für den Aufruf eines Flußdiagramm-Unterprogramms. Die Ausführung des aufrufenden FD-Programms wird unterbrochen, bis die Ausführung des Unterprogramms beendet ist.

Spezifische E/A-Aktionen im FD: Das Symbol einer spezifischen E/A-Aktion stellt auszuführende Aktionen dar. Wie alle Aktionen werden auch die spezifischen E/A-Aktion jeweils mit einer Nummer und einem Namen gekennzeichnet. Standardaktionen und spezifische E/A-Aktionen benutzen die gleiche Semantik. Die spezifischen E/A-Aktionen sollen das Flußdiagramm leserlicher machen und nicht-portierbare Abschnitte des Diagramms hervorheben. Die Verwendung von spezifischen E/A-Aktionen ist optionell. Spezifische E/A-Aktionsbausteine haben die gleichen Eigenschaften wie die Standardaktionen.

FD-Anschlüsse: Ein FD-Anschluß stellt die Verbindung von zwei Diagrammpunkten her, ohne daß die Verbindung gezeichnet werden muß. Ein solcher Anschluß wird als Kreis dargestellt und ist an die Quelle des Datenflusses angeschlossen. Je nach Richtung des Datenflusses wird das Anschlußsymbol an der entsprechenden Seite mit der Identifizierung des Zielpunkts vervollständigt (im allgemeinen der Name eines Zielsymbols). Das Ziel eines Anschlusses ist stets ein bereits definiertes FD-Symbol. Das Zielsymbol wird durch seine logische Nummer identifiziert.

FD-Kommentare: Die Kommentarfelder enthalten Text, der keinen Einfluß auf die Semantik des Flußdiagramms nimmt, und können an beliebigen freien Stellen im Dokumentfenster eines Flußdiagramms eingefügt werden. Kommentare dienen der Programmdokumentation.

A.5.2. Eingabe eines Flußdiagramms

Für die Erstellung eines Flußdiagramms werden die verschiedenen Elemente (Aktionen, Entscheidungstests, Anschlüsse...) in das grafische Eingabefeld eingefügt und Verbindungslinien zwischen den Elementen gezogen.

Objekte einfügen

Um ein Objekt in das Diagramm einzufügen, wählt man die entsprechende Schaltfläche in der Toolbox und klickt im grafischen Eingabefeld auf die Einfügeposition. Man kann das Element an einer freien Stelle einfügen oder auf einen Datenfluß setzen, indem man auf eine Datenflußverbindung klickt. Das ist jedoch nur bei vertikalen Verbindungen möglich, die von oben nach unten verlaufen.

Die folgenden Grundelemente können eingefügt werden:

 in ST, AWL oder im Schnellen KOP programmierte Aktion

 spezifische E/A-Aktion (hebt eine besondere nicht-portierbare Aktion hervor)

 in ST, AWL oder im Schnellen KOP programmierter Test (Entscheidung)

 Anschluß

 Aufruf eines FD-Unterprogramms

 Kommentar (Beschreibungstext)

Der ISaGRAF FD-Editor bietet auch eine Liste klassischer Flußdiagrammstrukturen. Solche Strukturen können nur auf eine bereits vorhandene Datenflußverbindung gesetzt werden, nicht in ein freies Feld:

 If / Then / Else (Binärauswahl)

 Repeat until (wartet auf eine Bedingung)

 While (Schleifenausführung, solange die Bedingung true ist)

Objekte auswählen

Für die meisten Bearbeitungsbefehle ist die Auswahl grafischer Objekte erforderlich. Mit dem ISaGRAF FD-Grafikeditor können ein oder mehrere im Diagrammfeld vorhandene Objekte ausgewählt werden. Um mehrere Objekte auszuwählen, wird die "**Auswahl**"-Schaltfläche (Pfeil) in der Editor-Toolbox aktiviert. Um ein einzelnes Objekt auszuwählen, klickt man einfach auf sein Symbol.
Für die Auswahl mehrerer Objekte wird die Maus im Diagramm "gezogen", wodurch sich ein Auswahlrechteck auf dem Bildschirm abzeichnet. Alle grafischen Objekte im Auswahlrechteck erscheinen nun als "**ausgewählt**".
Ausgewählte Objekte besitzen kleine schwarze Kästchen und heben sich durch ihre dunkelblaue Farbe von den restlichen Objekten ab. Wenn mehrere Objekte ausgewählt sind, kann ein Objekt hinzugefügt oder entfernt werden, indem man auf sein Symbol klickt, während man die **Shift**- oder **Strg**-Taste gedrückt hält.
Eine neue Auswahl hebt die bestehende Auswahl auf. Wenn die bestehende Auswahl aufgehoben werden soll, ohne eine neue Auswahl vorzunehmen, klickt man einfach mit der Maus auf eine freie Stelle außerhalb des Auswahlrechtecks.
Wenn nur ein Objekt ausgewählt ist, kann die Auswahl mit Hilfe der Pfeiltasten auf das nächste Objekt verlagert werden. Datenflußverbindungen können ebenfalls ausgewählt werden.

Kommentare einfügen

Kommentare können an beliebigen freien Stellen im Diagramm eingefügt werden. Sie nehmen keinen Einfluß auf die Programmausführung, verbessern jedoch die Leserlichkeit des Diagramms. Um ein Kommentarsymbol einzufügen, aktiviert man die entsprechende Schaltfläche in der Toolbox und klickt im Diagramm auf die gewünschte Einfügeposition. Doppelklicken Sie auf ein vorhandenes Kommentarsymbol, um Text einzugeben oder den vorhandenen Text zu modifizieren. Bei der Eingabe des Kommentartexts sind keine besonderen Kommentareinleitungs- oder -abschlußzeichen erforderlich. Wenn man die Ecken eines ausgewählten Kommentarsymbols mit der Maus zieht, verändert sich dessen Größe.

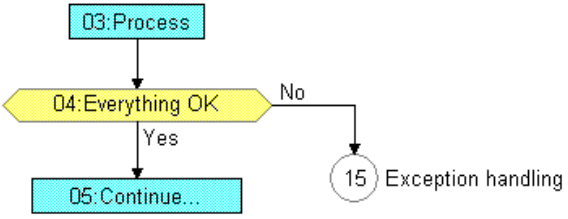
Datenflußverbindungen zeichnen

Aktivieren Sie diese Schaltfläche in der Toolbox, um eine Datenflußverbindung zu zeichnen. Die Verbindung muß in Richtung des Datenflusses verlaufen. Wählen Sie den freien Ausgangspunkt eines FD-Elements und ziehen Sie die Maus zum Zielpunkt, um die Verbindung anzuschließen. Der Zielpunkt kann entweder ein freier

Eingangspunkt (auf der oberen Seite eines FD-Elements) oder eine beliebige Stelle einer bestehenden Verbindung sein. Zusammenführungspunkte zwischen Verbindungen werden durch kleine graue Kreise im Datenflußdiagramm dargestellt. Solche Zusammenführungspunkte können ausgewählt und verschoben werden, um das Diagramm zu ordnen.

Anschlüsse benutzen

Im ISaGRAF Flußdiagramm-Editor können grafische Anschlüssen anstatt der sichtbaren Datenflußverbindungen benutzt werden, um sehr lange Verbindungslinien zu vermeiden und die Leserlichkeit des Diagramms zu verbessern. Diese Anschlüsse können jedoch nicht für die Verknüpfung mit anderen FD-Programmen verwendet werden.
Die Anschlüsse werden wie die anderen FD-Objekte in das Diagramm eingefügt. Sie werden durch einen Kreis dargestellt, der die numerische Referenz des Zielelements enthält (Ziel der Datenflußverbindung). Neben dem Anschlußkreis erscheint eine kurze Textbeschreibung des Zielelements.



Objekte verschieben

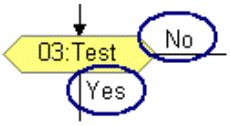
Wenn Objekte im Diagramm verschoben werden sollen, werden sie zunächst ausgewählt und dann mit der Maus im Diagramm "gezogen". Es können sowohl einzelne Elemente als auch multiple Auswahlen verschoben werden, wobei Überlappen der Elemente nicht zulässig ist. Diese Funktion kann nicht dazu benutzt werden, die Elemente an eine bestehende Datenflußverbindung anzuschließen.
Beim Verschieben eines einzelnen Elements (Aktion, Test...) werden alle unterhalb liegenden und an dieses Element angeschlossenen Elemente automatisch mit dem ausgewählten Element verschoben. Bei multiplen Auswahlen ist dies jedoch nicht der Fall.

Objektgröße verändern

Mit Ausnahme der "Begin"- und"End"-Symbole und der Anschlüsse sind alle grafischen Elemente eines Flußdiagramms in ihrer Größe veränderlich. Wenn die Größe eines Elements verändert werden soll, wählt man das Element aus und zieht mit der Maus an den kleinen Kästchen auf seinem Rand, bis man es auf die gewünschte Größe gebracht hat.
Wenn das Element an eine Datenflußverbindung angeschlossen ist und auf horizontaler Ebene vergrößert oder verkleinert wird, so geschieht dies nach beiden Seiten hin (nach links und nach rechts), so daß das Element weiterhin in seiner Mitte an die Verbindung angeschlossen ist.

Testausgänge austauschen

Die YES / NO Ausgänge eines Tests (Entscheidung) können ausgetauscht werden. Hierzu doppelklickt man einfach auf das "Yes" oder "No"-Symbol neben dem Testsymbol.



A.5.3. Ein bestehendes Netzwerk bearbeiten

Mit den Befehlen im Menü "**Bearbeiten**" können bestehende Diagramme verändert oder vervollständigt werden. Die meisten dieser Befehle betreffen die in einem Diagramm ausgewählten Elemente.

Netzwerkkorrekturen

Ausgewählte Elemente können mit der ENTF.-Taste gelöscht werden. Unvollständige Verbindungen werden mit den ausgewählten Elementen gelöscht. Benutzen Sie den Befehl "**Bearbeiten / Widerrufen**", um Elemente wiederherzustellen, die mit der ENTF.-Taste gelöscht wurden. Der Löschbefehl kann auch für eine Gruppe ausgewählter Elemente benutzt werden. Die Befehle "**Ausschneiden**", "**Kopieren**", "**Einfügen**" im Menü "**Bearbeiten**" werden benutzt, um die ausgewählten Elemente zu verschieben oder zu kopieren.

Suchen und Ersetzen

Die Befehle "**Bearbeiten / Suchen Ersetzen**" werden benutzt, um Text im gesamten Diagramm (in allen in ST, AWL oder im Schnellen KOP programmierten Aktionen und Tests) zu suchen und zu ersetzen. Der zu suchende Text wird im Dialogfeld Suchen/Ersetzen eingegeben. Wenn der gesuchte Text angetroffen wird, erscheint die entsprechende Stelle automatisch auf dem Bildschirm.

Direkter Zugriff auf ein Element

Der Befehl "**Bearbeiten / Gehen zu**" ermöglicht den schnellen Zugriff auf ein grafisches Element im Flußdiagramm. Der Bildablauf wird automatisch so ausgerichtet, daß das Element auf dem Bildschirm sichtbar wird. Das Element erscheint als "ausgewählt".

Elemente neu nummerieren

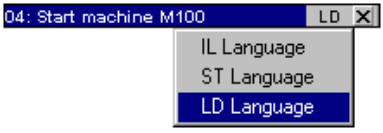
Mit dem Befehl "**Bearbeiten / Neu nummerieren**" können die Elemente eines Flußdiagramms neu nummeriert werden. Jedes der Elemente eines Flußdiagramms besitzt seine eigene Referenznummer, die beim Einfügen vom Editor zugeordnet wird. Mit dem Befehl "**Neu nummerieren**" kann die Nummerierung dieser Elemente entsprechend ihrer Positionen im Diagramm korrigiert werden. Die Nummerierung erfolgt von oben nach unten und von links nach rechts.

A.5.4. Programmierung der Ebene 2

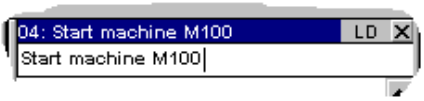
Die Texteingabe der Ebene 2 erfolgt durch Doppelklicken auf ein Aktions- oder Testsymbol. Das Programmierfeld der Ebene 2 erscheint rechts im FD-Fenster. Die Trennlinie zwischen FD-Diagramm und Ebene 2 kann beliebig verschoben werden. Es besteht die Möglichkeit, ein oder zwei Programmierfelder der Ebene 2 gleichzeitig zu öffnen. Die folgenden Befehle sind auf der Tastatur, mit der Maus oder im Menü "Bearbeiten" verfügbar:

Tastatur	Maus	Menü "Bearbeiten"
Eingabe	Doppelklicken	Ebene 2 bearbeiten
Strg+Eingabe	Strg + Doppelklicken	Ebene 2 in separatem Fenster bearbeiten

Wenn zwei Fenster der Ebene 2 sichtbar sind, kann die Trennline zwischen den beiden Fenstern beliebig verschoben werden. Die Schaltfläche rechts in der Titelzeile schließt das Fenster der Ebene 2.
Für die Ebene 2 wird **ST** (Strukturierter Text) als Programmiersprache vorgegeben, die Programmierung kann aber auch in **AWL** oder im **Schnellen KOP** erfolgen. Der Name der ausgewählten Sprache erscheint in einem kleinen Kästchen in der Titelzeile der Ebene 2. Leiten Sie den Menübefehl "**Optionen / Programmiersprache Ebene 2**" ein oder klicken Sie auf das Kästchen, um eine andere Sprache zu aktivieren. Dieser Befehl ist nur verfügbar, wenn das Fenster der Ebene 2 leer ist.



Ein einzeiliges Eingabefeld erscheint oberhalb des Fensters der Ebene 2. Es dient der Eingabe einer kurzen Beschreibung. Dieser Text wird als IEC-Kommentar in den Abbildungen von FD-Symbolen angezeigt. Er ist sehr nützlich, da er von bestimmten Befehlen wie zum Beispiel "Gehen zu..." benutzt wird und in der ausgedruckten Dokumentation die Aktionen und Tests beschreibt.

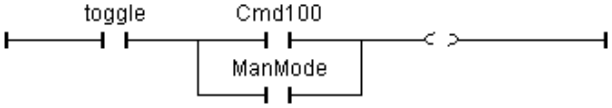


Der Befehl "**Optionen / Aktualisieren**" kann jederzeit benutzt werden, wenn Fenster der Ebene 2 offen sind, um das FD-Hauptdiagramm mit den modifizierten Programmen der Ebene 2 zu aktualisieren.

A.5.5. Programmierung der Ebene 2 im schnellen KOP

Die Ebene 2 kann auch mit dem Schnellen KOP-Editor programmiert werden. Im Fall eines Entscheidungstests besteht der Kontaktplan aus nur einem Rung, mit einer einzigen Spule, welche die Entscheidung darstellt. Der Name des Tests wird

nicht mit dem Spulensymbol wiederholt. Hier das Beispiel eines Tests, der im Schnellen KOP programmiert wurde.



Beim Programmieren im Schnellen KOP wird der Eingabecursor im logischen Programmierraster mit Hilfe der Pfeiltasten verlagert. Dann werden die Symbole eingesetzt, und zwar durch Anschlagen der folgenden Funktionstasten:
F2:.....fügt einen Kontakt hinter dem ausgewählten Symbol ein / initiiert das Rung
F3:.....fügt einen Kontakt vor dem ausgewählten Symbol ein
F4:.....fügt einen Kontakt parallel zum ausgewählten Symbol ein
F5:.....fügt eine Spule parallel zur ausgewählten Spule ein (nicht für Tests)
F6:.....fügt einen Baustein hinter dem ausgewählten Symbol ein
F7:.....fügt einen Baustein vor dem ausgewählten Symbol ein
F8:.....fügt einen Baustein parallel zum ausgewählten Symbol ein
F9:.....fügt ein Sprungsymbol parallel zur ausgewählten Spule ein (nicht für Tests)

Ein Sprung führt zu einem Rung-Namen. Der Name des Rung kann eingegeben werden, wenn die EINGABE-Taste gedrückt wird, während sich die Auswahl auf einem Runganfang befindet. Der ISaGRAF Editor speichert die eingegebenen Rung-Marken, egal ob diese für einen Rung-Namen oder eine Sprungoperation spezifiziert wurden. Im Dialogfeld "Sprung/Marke"' kann entweder eine neue Marke eingegeben oder eine bestehende Marke ausgewählt werden. Wenn ein neuer Name eingegeben wird, wird dieser automatisch der Liste beigefügt. Die Schaltfläche "**Entfernen**" wird benutzt, um den ausgewählten Namen aus der Liste zu nehmen. Die Marke auf dem im Diagramm ausgewählten Rung bleibt bestehen. Wenn diese Marke entfernt werden soll, klickt man einfach **OK**, wenn das Eingabefeld leer ist.
Anstatt der Funktionstasten können Sie auch die Schaltflächen in der KOP-Toolbox benutzen.
Drücken Sie die EINGABE-Taste, wenn sich der Eingabecursor auf einem Kontakt oder dem E/A-Parameter eines Bausteins befindet, um eine Variable auszuwählen oder einen konstanten Wert einzugeben. Drücken Sie die EINGABE-Taste, wenn sich der Eingabecursor auf einem Funktionsbaustein befindet, um den Bausteintyp auszuwählen. Doppelklicken auf ein Symbol bewirkt das gleiche.
Drücken Sie Strg + LEERTASTE, wenn ein Kontakt ausgewählt ist, um den Kontakt- oder Spulentyp (direkt oder invertiert) zu verändern. Siehe Kapitel "Schnellen KOP-Editor benutzen" in diesem Handbuch für weitere Informationen über die Kapazitäten des Schnellen KOP.

A.5.6. Anzeigeoptionen

Der Befehl "**Optionen / Layout**" öffnet ein Dialogfeld, in dem sämtliche Parameter und Optionen für die Gestaltung des Arbeitsbereichs und für das grafische Layout des Diagramms zusammengefaßt sind. Benutzen Sie die Kontrollkästchen im "Arbeitsbereich", um die Editor-Toolboxen und die Statuszeile anzuzeigen oder nicht anzuzeigen. Die Optionen im Feld "Dokument" aktivieren und deaktivieren die Anzeige des Bearbeitungsrasters und die Farbanzeige.

Benutzen Sie die "Zoom"-Schaltfläche in der Editor-Toolbox, um von einem zum anderen Vorgabefaktor zu wechseln. Dieser Befehl ist auch verfügbar, wenn Sie ein Schnelles KOP-Programm bearbeiten, das einer Aktion oder einem Test zugeordnet ist.

Mit der "Raster"-Schaltfläche können die Punkte des Bearbeitungsrasters angezeigt oder versteckt werden. Dieser Befehl ist auch verfügbar, wenn Sie ein Schnelles KOP-Programm bearbeiten, das einer Aktion oder einem Test zugeordnet ist.

Der Befehl "**Optionen / Schriftart**" dient der Auswahl der Schriftart für alle ISaGRAF Dokumente. Wenn dieser Befehl von einem ST- oder AWL-Block aus eingeleitet wird, können Sie auch die Größe der Schriftart bestimmen. Bei der Auswahl der Schriftart für die grafischen Abbildungen (LD oder Schneller KOP) brauchen Stil und Größe nicht bestimmt zu werden. Die ISaGRAF Grafikeditoren berechnen die Schriftgröße entspechend des aktuellen Zoomfaktors.

A.6. Schnellen KOP-Editor benutzen

Die KOP-Sprache ermöglicht die grafische Darstellung boolescher Ausdrücke. Die booleschen Operatoren AND, OR und NOT werden explizit durch die Diagrammtopologie dargestellt. Boolesche Eingangsvariablen werden mit grafischen Kontakten verbunden, boolesche Ausgangsvariablen mit grafischen Spulen. Der Schnelle KOP-Editor von ISaGRAF bietet eine vereinfachte Diagrammerstellung mit Hilfe der Tastatur oder der Maus: Die Elemente werden automatisch miteinander verbunden und auf Rungs angeordnet, so daß der Benutzer keine Verbindungslinien mehr zu ziehen braucht. Der Schnelle KOP-Editor ordnet die Rungs im Diagramm so an, daß der vom Diagramm eingenommene Platz optimal ausgenutzt wird.

A.6.1. Eigenschaften der KOP-Sprache

Ein KOP-Programm besteht aus einer Reihe von **Rungs**, auf denen Kontakte und Spulen angeordnet sind. Es folgen die grundlegenden Komponenten eines KOP-Diagramms:

Rung-Anfang (linke Stromschiene)

Jedes Rung beginnt mit einer linken Stromschiene, die den Initialzustand "TRUE" darstellt. Der Schnelle KOP-Editor von ISaGRAF erstellt die linke Stromschiene automatisch, wenn der erste Kontakt des Rung vom Benutzer eingesetzt wird. Jedes Rung kann einen logischen Namen besitzen, der als Marke für Sprunganweisungen benutzt werden kann.

Kontakte

Ein Kontakt modifiziert den booleschen Datenfluß entsprechend des Zustands einer booleschen Variablen. Der Name der Variablen erscheint über dem Kontaktsymbol. Die folgenden Kontakttypen werden vom Schnellen ISaGRAF KOP-Editor unterstützt:

 ..... direkter Kontakt

 ..... invertierter Kontakt

 ..... Kontakt mit Erkennung positiver (steigender) Flanke

 ..... Kontakt mit Erkennung negativer (fallender) Flanke

Spulen

Eine Spule stellt eine Aktion dar. Der Zustand des Rung (Zustand der linken Verbindung der Spule) wird benutzt, um eine boolesche Variable zu forcieren. Der Name der Variablen erscheint über dem Spulensymbol. Die folgenden Spulentypen werden vom Schnellen ISaGRAF KOP-Editor unterstützt:

 ..... direkte Spule

 ..... invertierte Spule

 ..... Spule mit "Setze"-Aktion

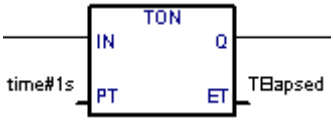
 ..... Spule mit "Rücksetze"-Aktion

 ..... Spule mit Erkennung positiver (steigender) Flanke

 ..... Spule mit Erkennung negativer (fallender) Flanke

Funktionsbausteine

Ein Funktionsbaustein in einem KOP-Diagramm kann eine Funktion, einen Funktionsbaustein, ein Unterprogramm oder einen Operator darstellen. Seine ersten Ein- und Ausgangsparameter sind immer an das Rung angeschlossen. Andere Ein- und Ausgangsparameter können außerhalb des Funktionsbaustein-Rechtecks geschrieben werden.



Rung-Ende (rechte Stromschiene)

Ein Rung endet in einer rechten Stromschiene. Der Schnelle KOP-Editor setzt automatisch eine rechte Stromschiene, wenn der Benutzer eine Spule einfügt.

Sprungsymbol

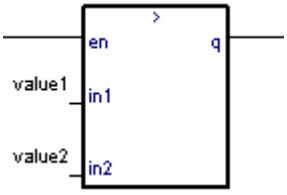
Ein Sprungsymbol referenziert immer eine Rung-Marke, d.h. den Namen eines Rung, der an einer anderen Stelle im KOP-Diagramm definert ist. Das Sprungsymbol wird an das Ende eines Rung plaziert. Wenn der Zustand des Rung TRUE ist, springt die Diagrammausführung direkt zum Ziel-Rung. Achtung: Rückwärtssprünge sind gefährlich, da sie die SPS-Zyklusschleife blockieren können.

Return-Symbol

Ein Return-Symbol wird an das Ende eines Rungs gesetzt. Es kennzeichnet, daß die Programmausführung gestoppt wird, wenn der Zustand des Rung TRUE ist.

Der "EN"-Eingang

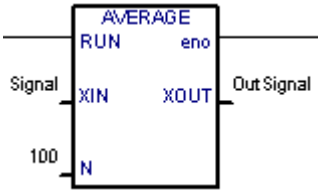
Bei manchen Operatoren, Funktionen und Funktionsbausteinen ist der erste Eingang nicht boolesch. Da der erste Eingang immer an das Rung angeschlossen werden muß, wird automatisch ein anderer Eingang namens "**EN**" an der ersten Stelle eingefügt. Der Baustein wird nur dann ausgeführt, wenn der **EN**-Eingang TRUE ist. Es folgt das Beispiel eines Vergleichsoperatoren, mit dem äquivalenten Code in ST:




```
IF rung_state THEN
   q := (wert1 > wert2);
ELSE
   q := FALSE;
END_IF;
(* rung mit q-zustand fortführen *)
```

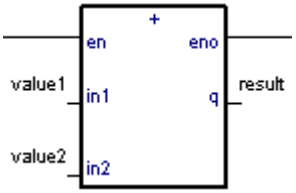
Der "ENO"-Ausgang

Bei manchen Operatoren, Funktionen und Funktionsbausteinen ist der erste Ausgang nicht boolesch. Da der erste Ausgang immer an das Rung angeschlossen werden muß, wird automatisch ein anderer Ausgang namens "**ENO**" an der ersten Stelle eingefügt. Der **ENO**-Ausgang nimmt immer den gleichen Zustand wie der erste Eingang des Bausteins an. Es folgt das Beispiel eines AVERAGE-Funktionsbausteins, mit dem äquivalenten Code in ST:




```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* rung mit eno-zustand
   fortführen *)
```

In manchen Fällen sind **EN** sowie **ENO** erforderlich. Es folgt das Beispiel eines arythmetischen Operatoren, mit dem äquivalenten Code in ST:




```
IF rung_state THEN
   Result := (wert1 + wert2);
END_IF;
Eno := rung_state;
(* rung mit eno-zustand
   fortführen *)
```

Begrenzungen des schnellen KOP-Editors

Mit dem Schnellen ISaGRAF KOP-Editor ist es nicht möglich, einen Rung auf der rechten Seite einer Spule fortzusetzen (andere Kontakte oder Spulen einzufügen). Wenn mehrere Ausgänge auf einem Rung erfolgen sollen, müssen die entsprechenden Spulen parallel gesetzt werden.

A.6.2. Eingabe eines KOP-Diagramms

Alle Bearbeitungsbefehle des Schnellen KOP-Editors können entweder mit der Tastatur oder mit der Maus eingeleitet werden.

Das Bearbeitungsraster

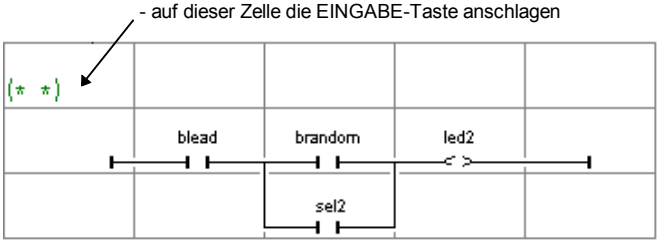
Das KOP-Diagramm wird in einer logischen Matrix eingegeben. Jede Matrixzelle enthält maximal ein KOP-Symbol. Benutzen Sie die Pfeiltasten oder klicken Sie auf eine Zelle, um die aktuelle Auswahl zu verlagern. Die ausgewählte Zelle erscheint im Inversvideo. Für manche Ausschneide-, Kopier- und Einfüge-Operationen ist es möglich, mehrere Zellen auszuwählen. Mit der Maus wird dies ganz einfach durch Ziehen des Mauscursors im Diagramm erreicht. Mit der Tastatur benutzen Sie die Pfeiltasten und halten dabei die SHIFT-Taste gedrückt.

Ein neues Rung starten

Um ein neues Rung in ein Diagramm einzufügen, verlagern Sie die Auswahl hinter das letzte Rung und fügen einen Kontakt ein (schlagen Sie F2 an oder klicken Sie auf die entsprechende Schaltfläche in der KOP-Toolbox). Sie erhalten ein neues Rung mit einem Kontakt und einer Spule.

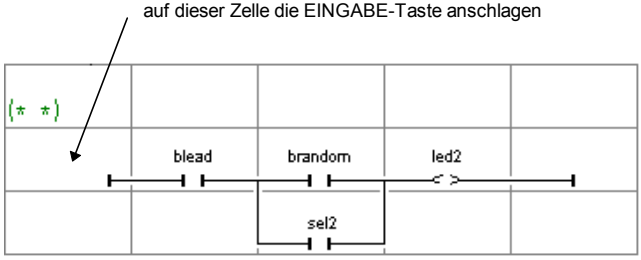
Eingabe des Rung-Kommentars

Jedes Rung kann mit bis zu zwei Textzeilen dokumentiert werden. Um den Kommentartext eines Rung einzugeben, verlagern Sie die Auswahl auf die Zelle über dem Rung und drücken die EINGABE-Taste. Sie können auch mit der Maus auf diese Zelle doppelklicken:



Eingabe der Rung-Marke

Jedes Rung kann mit einem Namen gekennzeichnet werden, der als Zielmarke für Sprungoperationen benutzt werden kann. Um die Marke eines Rung einzugeben oder zu ändern, verlagern Sie die Auswahl auf den Rung-Anfang und schlagen die EINGABE-Taste an. Sie können auch mit der Maus auf diese Zelle doppelklicken:



Der Schnelle ISaGRAF KOP-Editor speichert die bereits eingegebenen Rung-Marken, welche für Rung-Namen und Sprungoperationen definiert wurden.
Neu eingegebene Namen werden der Liste automatisch hinzugefügt. Die Schaltfläche "**Entfernen**" wird benutzt, um den ausgewählten Namen aus der Liste zu nehmen. Dieser Befehl entfernt nicht die Marke des im Diagramm ausgewählten Rung. Hierzu klicken Sie **OK**, wenn das Textfeld leer ist.

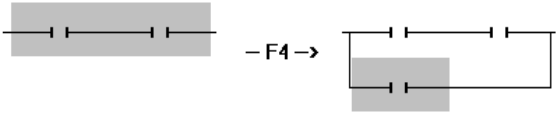
Symbole in ein Rung einfügen

Das Einfügen von Symbolen (Kontakten, Spulen, Funktionsbausteinen...) in ein bestehendes Rung erfolgt immer an der aktuellen Auswahlposition. Wählen Sie eine gültige Zellposition innerhalb des Rung und drücken Sie eine der folgenden Funktionstasten, um folgende Symbole einzufügen:

F2..... Kontakt vor dem ausgewählten Symbol (links)
F3..... Kontakt hinter dem ausgewählten Symbol (rechts)
F4..... Kontakt parallel zum ausgewählten Symbol
F6..... F-Baustein vor dem ausgewählten Symbol (links)
F7..... F-Baustein hinter dem ausgewählten Symbol (rechts)
F8..... F-Baustein parallel zum ausgewählten Symbol

Die folgenden Befehle sind gültig, wenn sich die Auswahl auf dem Rung-Ausgang (Spule) befindet
F5..... Spule parallel zum ausgewählten Symbol einfügen
F9..... "Sprung"-Symbol parallel zum ausgewählten Symbol einfügen
Shift + F9..... "Return"-Symbol parallel zum ausgewählten Symbol einfügen

Beim Einfügen paralleler Symbole (F4/F8), wenn mehrere Kontakte eines Rung gleichzeitig ausgewählt sind, wird das Symbol parallel zur ausgewählten Symbolgruppe eingefügt. Hier ein Beispiel:



Um Symbole in ein Diagramm einzufügen, können Sie auch die Befehle im Menü "**Hinzufügen**" benutzen. Sie können mit der Maus auf eine der Schaltflächen in der KOP-Toolbox klicken, um ein Symbol auszuwählen.

Symbole eingeben

Um einem Symbol (Kontakt oder Spule) eine Variable zuzuordnen, wählen Sie das betreffende Symbol aus und schlagen die EINGABE-Taste an. Sie können auch mit der Maus auf den Kontakt oder die Spule doppelklicken. Ein Dialogfeld für die Auswahl einer Variablen wird angezeigt. Siehe Kapitel "Mehr über die Programmeditoren" in diesem Dokument für weitere Informationen über die Benutzung dieses Dialogfelds. Um einem Baustein eine Funktion, einen Funktionsbaustein oder einen Operator zuzuordnen, schlagen Sie die EINGABE-Taste an, wenn sich die Auswahl innerhalb seines Rechtecks befindet. Um dem Ein- oder Ausgangsparameter eines Funktionsbausteins ein Variablensymbol zuzuordnen, muß sich die Auswahl an der betreffenden Stelle, **außerhalb** des Baustein-Rechtecks, befinden.
Gewöhnlich werden Dialogfelder mit Listenfeldern für die Auswahl von Variablen oder Funktionsbausteinen für die Texteingabe benutzt. Wenn die "**manuelle Eingabe**" im Menü "**Optionen**" aktiviert wird, werden die Variablensymbole und Bausteinnamen mit Hilfe der Tastatur direkt in ein einfaches Eingabefeld geschrieben. Geben Sie den neuen Text ein und drücken Sie die "**Eingabe**"-Taste, um die Eingabe zu validieren. Wenn Sie die "**Escape**"-Taste drücken, finden keine Änderungen statt und das Eingabefeld wird geschlossen. Das Eingabefeld für die "manuelle Eingabe" kann nicht mit der Maus geschlossen werden.

Kontakt- und Spulentyp verändern

Der Befehl "**Bearbeiten / Kontakt-/Spulentyp**" wird benutzt, um den Typ eines Kontakts oder einer Spule zu ändern. Kontakte können direkt, invertiert, oder mit Erkennung einer negativen oder positiven Flanke sein. Spulen können direkt,

invertiert, Setze, Rücksetze oder mit Erkennung einer negativen oder positiven Flanke sein. Dieser Befehl kann auch mit der LEERTASTE eingeleitet werden.

Ein Rung in ein Diagramm einfügen

Der Befehl "**Bearbeiten / Rung einfügen**" wird benutzt, um ein neues Rung vor dem im Diagramm ausgewählten Rung einzufügen. Das Rung wird mit einem Kontakt und einer Spule initialisiert.

A.6.3. Ein bestehendes Diagramm bearbeiten

Die Befehle im Menü "**Bearbeiten**" werden benutzt, um ein bestehendes Diagramm zu ändern oder zu vervollständigen. Die meisten dieser Befehle bearbeiten die im Diagramm ausgewählten Elemente.

Ein Diagramm korrigieren

Ausgewählte Elemente können mit der ENTF-Taste gelöscht werden. Es ist nicht möglich, ein Spulen-, Sprung- oder Return-Symbol zu löschen, wenn es der einzige Ausgang eines Rung ist. Benutzen Sie den Befehl "**Bearbeiten / Widerrufen**", um Elemente nach einem Löschbefehl wiederherzustellen. Der ENTF-Befehl kann auch für eine im Diagramm ausgewählte Gruppe von Elementen benutzt werden. Wenn sich die Auswahl auf dem Rung-Kommentartext befindet, kann der ENTF-Befehl benutzt werden, um den Text neu einzugeben. Wenn sich die Auswahl auf einem Rung-Anfang befindet, löscht der ENTF-Befehl das gesamte Rung.

Symbole kopieren

Die Befehle "**Ausschneiden**", "**Kopieren**"und "**Einfügen**" im Menü "**Bearbeiten**" werden benutzt, um ausgewählte Elemente zu verlagern oder zu kopieren. Diese Befehle nehmen keinen Einfluß auf die Rung-Kommentare. Der Befehl "**Bearbeiten / spezielles Einfügen**" ermöglicht, Elemente an bestimmten Stellen einzufügen:

- vor dem ausgewählten Element (links)
- hinter dem ausgewählten Element (rechts)
- parallel zum ausgewählten Element

Ganze Rungs verwalten

Wenn der Rung-Anfang (linke Stromschiene) ausgewählt ist, beziehen sich alle Bearbeitungsbefehle (löschen, kopieren, ausschneiden...) auf das gesamte Rung. Man braucht die Auswahl nur in die erste Spalte zu verlagern, um Rungs problemlos im Diagramm zu ordnen. Es ist auch möglich, die Auswahl senkrecht zu erweitern, so daß sie mehrere Rung-Anfänge einschließt. In diesem Fall können die Bearbeitungsbefehle auf eine ganze Rung-Liste angewendet werden.

Suchen und Ersetzen

Die Befehle "**Bearbeiten / Suchen**" und "**Bearbeiten / Ersetzen**" werden benutzt, um Texte im Diagramm zu suchen und zu ersetzen. Nur ganze Namen können gesucht werden. Der Suchbefehl bezieht sich auf Kontakte, Spulen, Funktionsbausteinnamen und -parameter und Ausführungsmarken. Er kann nicht benutzt werden, um eine Zeichenkette in einem Rung-Kommentar zu suchen. Der Befehl "Ersetzen" kann nicht benutzt werden, um den Typ eines Bausteins zu ändern. Die Suche kann vorwärts oder rückwärts verlaufen und beginnt am Standort

der aktuellen Auswahl. Wenn das Ende des Diagramms erreicht ist, geht die Suche am Anfang weiter und kehrt zum Ausgangspunkt zurück. Die folgenden Tastenkombinationen sind für die schnelle Suche eines Variablennamens verfügbar:
ALT + F2 sucht das nächste Element mit dem gleichen Variablennamen wie das ausgewählte Element. Diese Funktion kann auch bei Funktionsbausteinen und Rung-Marken angewendet werden.
ALT + F5 sucht die nächste Spule mit dem gleichen Variablennamen wie das ausgewählte Element. Diese Funktion wird hauptsächlich im Debugging-Modus benutzt, um schnell den Rung zu finden, der eine suspekte Variable forciert.

A.6.4. Anzeigeoptionen

Die Befehle im Menü "**Optionen**" werden benutzt, um die Zeichnung eines KOP-Diagramms auf dem Bildschirm persönlich zu gestalten und um bestimmte Informationen anzuzeigen oder zu verstecken.

Rung-Kommentare

Benutzen Sie den Befehl "**Optionen / Rung-Kommentare**", um die Rung-Kommentare des gesamten Diagramms anzuzeigen oder zu verstecken. Bei umfangreichen Diagrammen kann es von Vorteil sein, die Kommentare nicht anzuzeigen, um einen besseren Überblick über das Diagramm zu erhalten, da jeder Kommentar eine Zeile in der Bearbeitungsmatrix belegt. Diese Option nimmt keinen Einfluß auf den Inhalt der bestehenden Rung-Kommentare und kann jederzeit umgeschaltet werden.

Name und Alias

Jede Variable, die einem Kontakt, einer Spule oder dem E/A-Parameter eines Funktionsbausteins zugeordnet ist, wird mit einem symbolischen Namen gekennzeichnet. Der Schnelle ISaGRAF KOP-Editor benutzt außerdem einen "**Alias**" für jede Variable. Der Alias einer Variablen ist ihr Kommentartext, der vor dem ersten '**:**'-Zeichen abgekürzt ist und sich auf 16 Zeichen begrenzt. Hier einige Beispiele:

```
Variablenkommentar:              Alias:
short text                       short text
long text with no separator      long text with n
short text: long description     short text
```

Ein Alias nimmt keinen Einfluß auf die Ausführung des KOP-Diagramms und soll als Syntaxkommentar verstanden werden. Er wird automatisch aus dem Kommentar einer Variablen erstellt, wenn der Name in der Variablenliste ausgewählt wird, und kann nicht manuell geändert werden. Benutzen Sie den Befehl "**Optionen / Kontakte und Spulen**", um einen Anzeigemodus für die Variablen-Identifizierung auszuwählen. Folgende Modi sind verfügbar:

- Anzeige nur mit Variablennamen
- Anzeige nur mit Variablen-Alias
- Anzeige mit Namen und Alias

Der Schnelle KOP-Editor aktualisiert KOP-Dokumente nicht automatisch, wenn ein Variablenalias im Datenverzeichnis geändert wird. Benutzen Sie den Befehl

"Optionen / Kontakte und Spulen / Alias aktualisieren" um alle Aliasnamen des bearbeiteten Diagramms zu aktualisieren. Sie können auch die Option "Immer bei Öffnen aktualisieren" im Menü "Optionen / Kontakte und Spulen" aktivieren, damit ISaGRAF alle verwendeten Aliasnamen beim Öffnen eines Schnellen KOP-Programms aktualisiert. Achtung: Wenn diese Option aktiv ist, kann das Öffnen eines Programms wesentlich länger dauern.

Zeichenoptionen

Der Befehl "**Optionen / Layout**" öffnet ein Dialogfeld, in dem sämtliche Parameter und Optionen für die Gestaltung des Arbeitsbereichs und für das Layout des grafischen KOP-Diagramms zusammengefaßt sind. Benutzen Sie die Kontrollkästchen im "Arbeitsbereich", um Editor-Toolbox, Statuszeile und KOP-Toolbox anzuzeigen oder nicht anzuzeigen. Die Optionen im Feld "Dokument" aktivieren und deaktivieren die Anzeige des Bearbeitungsrasters und die Farbanzeige.

Die Optionen in der Gruppe "Zoom" ermöglichen die Auswahl des Zoomfaktors. Sie können auch die Schaltfläche "Zoom" in der Editor-Toolbox benutzen, um von einem zum anderen Vorgabefaktoren zu wechseln.

Auch der X/Y-Quotient der Rasterzellen kann verändert werden. Wenn Sie gewöhnlich kurze Variablennamen benutzen, ist es vorteilhaft, die vorgegebene Zellenbreite zu verringern. Sie können hierzu auch einfach auf die Schaltfläche "Breite" in der Editor-Toolbox klicken.

Der Befehl "**Optionen / Schriftart**" dient der Auswahl einer Schriftart für alle grafischen ISaGRAF Dokumente. Bei der Auswahl der Schriftart brauchen Stil und Größe nicht bestimmt zu werden. Die ISaGRAF Grafikeditoren berechnen die Größe der Schriftart gemäß des ausgewählten Zoomfaktors.

A.7. FBS/KOP-Editor benutzen

Der grafische FBS/KOP-Editor von ISaGRAF ermöglicht die Eingabe von kompletten FBS-Programmen, die Abschnitte in der KOP-Sprache enthalten können. Er kombiniert Grafik- mit Textbearbeitung, so daß nicht nur die Diagramme, sondern auch die entsprechenden Ein- und Ausgänge eingegeben werden können. Da dieser Editor hauptsächlich der FBS-Sprache gewidmet ist, sollten reine KOP-Diagramme mit dem Schnellen KOP-Editor von ISaGRAF erstellt werden.

A.7.1. Eigenschaften der FBS- und KOP-Sprachen

Die **FBS**-Sprache ist eine grafische Darstellung vieler unterschiedlicher Arten von Gleichungen. Die **Operatoren** werden durch rechteckige Funktionsbausteine dargestellt. Funktionseingänge werden auf der linken Seite eines Bausteins angeschlossen, Funktionsausgänge auf der rechten Seite. Diagrammeingänge und -ausgänge (**Variablen**) werden mit Hilfe von **logischen Verbindungen** an die Funktionsbausteine angeschlossen. Der Ausgang eines Funktionsbausteins kann an den Eingang eines anderen Funktionsbausteins angeschlossen werden.
Die **KOP**-Sprache ermöglicht die grafische Darstellung boolescher Ausdrücke. Die booleschen Operatoren **AND, OR** und **NOT** werden explizit durch die Diagrammtopologie dargestellt. Boolesche Eingangsvariablen werden grafischen **Kontakten** zugeordnet, boolesche Ausgangsvariablen grafischen **Spulen**. Kontakte und Spulen werden miteinander und mit den linken und rechten Stromschienen durch **horizontale Linien** verbunden. Jeder Abschnitt einer Linie hat einen booleschen Zustand **FALSE** oder **TRUE**. Der boolesche Zustand ist der gleiche für alle direkt miteinander verbundenen Abschnitte. Eine horizontale Verbindungslinie, die mit der linken **vertikalen Stromschiene** verbunden ist, hat den Zustand **TRUE**.
KOP und FBS-Diagramme werden immer von links nach rechts und von oben nach unten interpretiert. Siehe ISaGRAF Sprachreferenzen für eine ausführlichere Beschreibung der KOP- und FBS-Sprachen. Es folgen die grundlegenden grafischen Komponenten der KOP- und FBS-Sprachen, die vom FBS/KOP-Editor unterstützt werden:

Linke Stromschiene

Rungs müssen links an die **linke Stromschiene** angeschlossen werden, die den Initialzustand "TRUE" darstellt. Der ISaGRAF FBS-Editor ermöglicht außerdem, ein beliebiges boolesches Symbol an die linke Stromschiene anzuschließen.

Rechte Stromschiene

Spulen können rechts an die **rechte Stromschiene** angeschlossen werden. Im ISaGRAF FBS/KOP-Editor ist diese Funktionalität optionell. Wenn eine Spule rechts nicht angeschlossen wird, enthält sie eine rechte Stromschiene in ihrer Abbildung.

Vertikale "ODER"-Verknüpfung im KOP

Vertikale KOP-Verbindungen akzeptieren mehrere Anschlüsse links und mehrere Anschlüsse rechts. Jede rechte Verbindung gleicht der ODER-Kombination der linken Verbindungen.

Kontakte

Ein Kontakt modifiziert den booleschen Datenfluß entsprechend des Zustands einer booleschen Variablen. Der Name der Variablen erscheint über dem Kontaktsymbol. Die folgenden Kontaktypen werden vom ISaGRAF FBS/KOP-Editor unterstützt:

- ..... direkter Kontakt
- ..... invertierter Kontakt
- ..... Kontakt mit Erkennung positiver (steigender) Flanke
- ..... Kontakt mit Erkennung negativer (fallender) Flanke

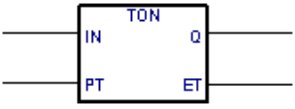
Spulen

Eine Spule stellt eine Aktion dar. Sie muß links an ein boolesches Symbol, z.B. einen Kontakt, angeschlossen werden. Der Name der Variablen erscheint über dem Spulensymbol. Die folgenden Spulentypen werden vom ISaGRAF FBS/KOP-Editor unterstützt:

- ..... direkte Spule
- ..... invertierte Spule
- ..... Spule mit "Setze"-Aktion
- ..... Spule mit "Rücksetze"-Aktion

Funktionsbausteine

Die Bausteine eines FBS-Diagramms können Funktionen, Funktionsbausteine, Unterprogramme oder Operatoren darstellen. Ein- und Ausgänge müssen an Variablen, Kontakte oder Spulen, oder an die Ein- oder Ausgänge anderer Bausteine angeschlossen werden. Die formalen Parameternamen werden innerhalb des Bausteinrechtecks angezeigt.



Marken

Marken können überall im Diagramm eingesetzt werden. Sie werden als Ziele für Sprunganweisungen benutzt, um die Ausführungsordnung des Diagramms zu modifizieren. Marken werden nicht an andere Elemente angeschlossen. Sie sollten auf die linke Seite des Diagramms gesetzt werden, um die Leserlichkeit des Diagramms zu verbessern.

Sprünge

Ein Sprungsymbol bezieht sich stets auf eine Marke, die sich an einer anderen Stelle im Diagramm befindet. Sein linker Anschluß muß mit einem booleschen Punkt verbunden werden. Wenn der linke Anschluß TRUE ist, springt die Diagrammausführung direkt zur betreffenden Zielmarke. Achtung : Rückwärtssprünge sind gefährlich, da sie eine Blockierung der SPS-Zyklusschleife verursachen können.

Return-Symbol

Ein Return-Symbol wird an einen booleschen Punkt angeschlossen. Es kennzeichnet, daß die Programmausführung gestoppt wird, wenn der Zustand des Rung TRUE ist.

Variablen

Variablen werden im Diagramm in kleinen Rechtecken dargestellt, die links oder rechts an andere Diagrammelemente angeschlossen sind.

Verbindungslinien

Nach Einsetzen der Elemente in das Diagramm werden Verbindungslinien zwischen den Elementen gezogen. Die Linien werden stets von einem Ausgangspunkt zu einem Eingangspunkt gezogen (in Richtung des Datenflusses).

Verbindungen mit Boolescher Negation

Manche booleschen Verbindungslinien werden mit einem kleinen Kreis an ihrem rechten Ende dargestellt. Dies bedeutet eine boolesche Negation der von der Verbindung transportierten Daten.

Benutzerdefinierte Ecken

Auf den Verbindungen können benutzerdefinierte Punkte (Ecken) definiert werden. Sie räumen dem Benutzer die Möglichkeit ein, den Verlauf einer Verbindung manuell zu steuern. Wenn keine Ecke gesetzt wird, benutzt der ISaGRAF FBS/KOP-Editor den vorgegebenen Algorythmus für den Verlauf der Verbindung.

A.7.2. Eingabe eines FBS-Diagramms

Um ein Diagramm einzugeben, werden die jeweiligen Elemente (Bausteine, Variablen, Kontakte, Spulen...) in das grafische Bearbeitungsfeld eingesetzt und die Verbindungen zwischen den Elementen gezogen.

Objekte einfügen

Um ein Objekt in ein Diagramm einzufügen, wählen Sie die entsprechende Schaltfläche in der Toolbox und klicken im grafischen Bearbeitungsfeld auf die Position, an der das Objekt eingefügt werden soll.

Objekte auswählen

Die meisten Bearbeitungsbefehle erfordern die Auswahl grafischer Objekte. Der grafische ISaGRAF KOP/FBS-Editor ermöglicht die Auswahl eines oder mehrerer Objekte im Diagramm. Für die Objektauswahl muß die "**Auswahl**"-Schaltfläche (Pfeil) in der Editor-Toolbox aktiviert werden. Um ein einzelnes Objekt auszuwählen, klickt man einfach auf sein Symbol. Um mehrere Objekte auszuwählen, zieht man die Maus im Diagrammfeld und erstellt ein Auswahlrechteck. Alle grafischen Objekte, die vom Auswahlrechteck durchkreuzt werden, erscheinen nun als "**ausgewählt**" (mit kleinen, schwarzen Kästchen auf ihren Rahmen). Eine neue Auswahl hebt die vorherige Auswahl auf. Um eine bestehende Auswahl zu entfernen, klickt man einfach mit der Maus in ein leeres Feld außerhalb des Auswahlrechtecks.

Kommentare einfügen

Kommentare können an beliebigen Stellen im Diagramm eingefügt werden. Sie nehmen keinen Einfluß auf die Programmausführung, verbessern jedoch die Leserlichkeit des Diagramms. Um einen Kommentarblock einzufügen, wählt man die entsprechende Schaltfläche in der Toolbox und zeichnet durch Ziehen mit der Maus ein rechteckiges Kommentarfeld. Nun kann der Kommentartext eingegeben werden. Spezialzeichen zur Begrenzung des Kommentartexts, wie "**(***" und "***)**", sind nicht erforderlich. Ein ausgewählter Kommentarblock kann durch Ziehen seiner Ecken vergrößert oder verkleinert werden.

Objekte verschieben

Um Objekte im Diagramm zu verschieben, werden die Objekte zuerst ausgewählt und dann mit der Maus an eine andere Stelle im Diagramm gezogen. Um miteinander verbundene Objekte zu verschieben, werden lediglich die entsprechenden grafischen Symbole verschoben. Die Verbindungslinien zwischen den verschobenen Objekten werden automatisch und unter Berücksichtigung ihrer neuen Lage vom ISaGRAF KOP/FBS-Editor erstellt.

Verbindungslinien zeichnen

Wählen Sie eine dieser Schaltflächen in der Toolbox, um eine Verbindung zwischen den Anschlußpunkten bestehender Elemente zu zeichnen. Wenn Sie eine Verbindung von einem Anschlußpunkt zu einer leeren Stelle im Diagramm ziehen, wird die Verbindung automatisch mit einer benutzerdefinierten Ecke versehen, so daß Sie gleich ein weiteres Segment zeichnen können.

Verlauf einer Verbindungslinie verändern

Der Befehl "**Werkzeuge / Linie verschieben**" wird benutzt, wenn der vorgegebene Verlauf einer im Diagramm ausgewählten Verbindungslinie geändert werden soll. Dieser Befehl ist wirkungslos, wenn die Verbindungslinie an eine benutzerdefinierte Ecke angeschlossen ist. Wenn eine Linie in drei Segmenten verläuft, ändert dieser Befehl die Position des zweiten Segments. Hier zwei Beispiele:

Typ einer Verbindungslinie verändern

Sie können den Typ einer Verbindungslinie (mit oder ohne boolescher Negation) ganz einfach ändern, indem Sie mit der Maus auf das rechte Ende der Linie doppelklicken.

KOP-Rungs zeichnen

Um ein neues KOP-Rung zu erstellen, setzen Sie zuerst die linke Stromschiene. Fügen Sie nun eine Spule ein: Die Spule wird automatisch mit der linken Stromschiene verbunden. Kontakte und vertikale ODER-Verknüpfungen können unmittelbar auf die Rung-Linie gesetzt werden, ohne neue Verbindungslinien zu zeichnen.

Wenn Sie ein neues KOP-Element (Kontakt oder Spule) an einer freien Stelle im Bearbeitungsfeld einfügen, wird automatisch eine neue horizontale Rung-Linie vom eingesetzten Element zu den bestehenden linken und rechten Stromschienen gezogen. Dies ist nicht der Fall, wenn das Element nicht zwischen Stromschienen eingefügt wird. Das eingefügte Element kann nun beliebig auf der neuen Rung-Linie verschoben werden. Die automatisch vom KOP-Editor erstellten, horizontalen Linien können ausgewählt und gelöscht werden. Es können auch neue Kontakt- oder Spulensymbole auf den horizontalen Linien eines bestehenden Rung eingefügt werden. Der Editor unterbricht das Rung automatisch und verbindet es mit den linken und rechten Anschlußpunkten des neu eingefügten Symbols.

Mehrfachverbindungen

Eine Mehrfachverbindung kann auf der rechten Seite eines beliebigen **Ausgangs**punktes erstellt werden. Dies bedeutet, daß die Daten zu mehreren anderen Punkten im Diagramm **gesendet** werden. Der gleiche Zustand wird in jedes rechte Ende propagiert. Die Anzahl der Verbindungslinien auf der rechten Seite eines Ausgangspunkts ist unbegrenzt. Die rechten Enden zweier Verbindungslinien können nicht an den gleichen **Eingang**spunkt angeschlossen werden, außer bei den folgenden KOP-Symbolen:

 ..... rechte Stromschiene

 ..... Mehrfachverbindung links (ODER-Operator)

Diese KOP-Symbole können eine unbegrenzte Anzahl an Eingängen besitzen.

A.7.3. Ein bestehendes Diagramm bearbeiten

Die Befehle im Menü "**Bearbeiten**" werden benutzt, um ein bestehendes Diagramm zu ändern oder zu vervollständigen. Die meisten dieser Befehle bearbeiten die im Diagramm ausgewählten Elemente.

Ein Diagramm korrigieren

Ausgewählte Elemente können mit der ENTF-Taste entfernt werden. Unvollständige Verbindungen werden mit den Elementen gelöscht. Benutzen Sie den Befehl "**Bearbeiten / Widerrufen**", um die gelöschten Elemente nach einem ENTF-Befehl widerherzustellen. Der ENTF-Befehl kann auch auf eine im Diagramm ausgewählte Gruppe von Elementen angewendet werden. Die Befehle "**Ausschneiden**", "**Kopieren**" und "**Einfügen**" im Menü "**Bearbeiten**" werden benutzt, um ausgewählte Elemente zu verschieben oder zu kopieren.

Suchen und Ersetzen

Die Menübefehle "**Bearbeiten / Suchen**" und "**Bearbeiten / Ersetzen**" werden benutzt, um Texte im Diagramm zu suchen und zu ersetzen. Nur ganze Namen können gesucht werden. Der Suchbefehl bezieht sich auf Kontakte, Spulen, Bausteinnamen, Variablen und Marken. Er kann nicht benutzt werden, um eine Zeichenkette in einem Kommentartext zu suchen. Der Befehl "Ersetzen" kann nicht benutzt werden, um den Namen eines Bausteins zu ändern. Die Suche kann vorwärts oder rückwärts verlaufen und beginnt am Standort der aktuellen Auswahl. Wenn das Ende des Diagramms erreicht ist, geht die Suche am Anfang weiter, zurück zum Ausgangspunkt.

Ausführungsordnung anzeigen

Wenn ein FBS-Diagramm Rückwärtsschleifen einschließt, ist die normale Ausführungsordnung "von links nach rechts / von oben nach unten" nicht mehr gegeben. Um Verwechselungen zu vermeiden, benutzen Sie den Befehl "**Werkzeuge / Ausführungsordnung zeigen**" oder die Tastenkombination **Strg + F1**, um die Ausführungsordnung anzuzeigen, die bei der Kompilierung benutzt wird. Identifizierungsnummern von 1 bis N werden neben den Symbolen, die zu einer Aktion führen (Spulen, Setze-Variablen und Funktionsbausteine), angezeigt.

Symbole und Text eingeben

Doppelklicken Sie mit der Maus auf ein Element, um das zugeordnete Symbol oder den zugeordneten Text einzugeben. Dies betrifft Variablen, Kontakte und Spulen, Kommentartexte und Marken. Bei Kontakten und Spulen kann auch der Typ geändert werden (direkt, invertiert...).
Die Texteingabe erfolgt gewöhnlich in Dialogfeldern, die über Listenfelder für die Auswahl von Variablen oder Bausteinen verfügen. Wenn "**Manuelle Eingabe**" im Menü "**Optionen**" aktiviert wird, können die Variablensymbole und Bausteinnamen unter Anwendung der Tastatur unmittelbar in ein einfaches Textfeld geschrieben werden. Der eingegebene Text wird mit der "**Eingabe**"-Taste validiert. Wenn Sie die "**Escape**"-Taste drücken, wird die Eingabe nicht berücksichtigt und das Textfeld geschlossen. Das Textfeld für die "manuelle Eingabe" kann nicht mit der Maus geschlossen werden.
Wenn "**Autm. Eingabe**" im Menü "**Optionen**" aktiviert wird, muß das Variablensymbol sofort nach Einfügen eines neuen Kontakts oder einer neuen Spule eingegeben werden. Wenn eine Variable oder eine Marke eingefügt wird, muß das Symbol immer sofort eingegeben werden.

Funktionsbausteintyp auswählen

Doppelklicken Sie mit der Maus auf einen Funktionsbaustein, um seinen Typ zu ändern. Der Bausteintyp wird aus einer Liste verfügbarer Operatoren, Funktionen und Funktionsbausteine ausgewählt. Mit diesem Befehl kann auch die Anzahl der Eingangspunkte eines kommutativen Operators geändert werden. (z.B. AND, OR, ADD, MUL...)

Freier Platz

Wenn sich der Cursor im FBS-Zeichenfeld befindet, können Sie durch Drücken der rechten Maustaste ein Popup-Menü anzeigen. Es enthält die folgenden Befehle, die benutzt werden können, um freien Platz an der Position des Cursors einzufügen oder zu entfernen:

Reihen einfügen: Dieser Befehl fügt auf horizontaler Ebene 4 Reihen freien Platz ein (entsprechend der Rasterstufen), angefangen an der Position des Cursors, wo das Popup-Menü geöffnet wurde.

Reihen löschen: Dieser Befehl entfernt auf horizontaler Ebene unbenutzten Platz (Reihen), angefangen an der Position des Cursors, wo das Popup-Menü geöffnet wurde. Dieser Befehl kann nicht benutzt werden, um FBS-Elemente zu entfernen.

Wenn das Popup-Menü offen ist, kennzeichnet eine graue Linie im FBS-Zeichenfeld, wo freier Platz eingefügt oder gelöscht wird.

A.7.4. Anzeigeoptionen

Die Befehle im Menü "**Optionen**" werden benutzt, um das Zeichnen eines FBS-Diagramms auf dem Bildschirm persönlich zu gestalten.

Persönliche Gestaltung des Layouts

Der Befehl "**Optionen / Layout**" öffnet ein Dialogfeld, in dem sämtliche Parameter und Optionen für die Gestaltung des Arbeitsbereichs und für das Layout des grafischen Diagramms zusammengefaßt sind. Benutzen Sie die Kontrollkästchen im "Arbeitsbereich", um die Editor-Toolboxen und die Statuszeile anzuzeigen oder zu verstecken. Die Optionen im Feld "Dokument" aktivieren und deaktivieren die Anzeige des Bearbeitungsrasters.

Die Optionen in der Gruppe "Zoom" ermöglichen die Auswahl des Zoomfaktors. Sie können auch die Schaltfläche "Zoom" in der Editor-Toolbox benutzen, um von einem zum anderen Vorgabefaktoren zu wechseln.

Der Befehl "Optionen / Schriftart" dient der Auswahl der Schriftart für alle grafischen ISaGRAF Dokumente. Stil und Größe der Schriftart braucht nicht definiert zu werden. Die Schriftgröße wird von den ISaGRAF Grafikeditoren automatisch entsprechend des Zoomfaktors berechnet.

A.7.5. Stilarten und Anzeige der Änderungen

Mit dem ISaGRAF KOP/FBS-Editor kann einem beliebigen Element eines KOP/FBS-Diagramms ein bestimmter grafischer Stil zugeordnet werden. Dabei handelt es sich hauptsächlich um eine bestimmte Farbgebung. Es werden aber auch Stilarten benutzt, um die Änderungen beim Vergleich von zwei unterschiedlichen Versionen eines Diagramms hervorzuheben.
Hinweis: Beim Debuggen im Simulations- oder Onlinemodus sind die Stilarten nicht sichtbar, da die Farben rot und blau in diesem Modus die TRUE / FALSE Zustände der beobachteten Variablen kennzeichnen.

Festgelegte Stilarten

Folgende Stilarten sind festgelegt:

Normal..... Default-Zeichnung (schwarz). Bei der Anzeige der Änderungen kennzeichnet der "normale" Stil, daß diese Elemente zum ursprünglichen Diagramm gehören. Elemente, die im "normalen" Stil erscheinen, werden bei der Diagrammausführung normal durchlaufen.

Modifiziert ..... Als "modifiziert" gekennzeichnete Elemente erscheinen in rosa. Dieser Stil wird bei der Anzeige der Änderungen benutzt, um Elemente hervorzuheben, die im Ursprungsdiagramm geändert oder hinzugefügt wurden. Elemente, die im "modifizierten" Stil erscheinen, werden bei der Programmausführung normal durchlaufen.

Gelöscht..... "Gelöschte" Elemente erscheinen grau gestrichelt. Solche Elemente werden bei der Diagrammausführung nicht berücksichtigt. Dieser

Stil wird benutzt, um anzuzeigen, welche Elemente aus der Ursprungsversion des Diagramms entfernt wurden, wenn die alte mit der neuen Version verglichen werden sollen.

Benutzer..... Zusätzlich zu den vordefinierten Stilarten ermöglicht der ISaGRAF KOP/FBS-Editor die Auswahl einer beliebigen Farbe für einen bestimmten Diagrammabschnitt. Dieser vom "Benutzer" definierte Stil nimmt keinen Einfluß auf die Diagrammausführung.

Benutzen Sie die Befehle des Untermenüs "**Stil**" im Menü "**Bearbeiten**", um den ausgewählten Elementen manuell einen bestimmten Stil zuzuordnen.

Anzeige der Änderungen

Die Anzeige der verschiedenen Stilarten und die Verfügbarkeit des Stils "Gelöscht" ermöglichen dem Benutzer, die Änderungen eines bestehenden Diagramms zu verfolgen. Benutzen Sie den Befehl "**Änderungen markieren**" im Menü "**Bearbeiten/Stil**", um die Anzeige der Änderungen zu aktivieren oder zu unterbinden.

Wenn die Option "Änderungen markieren" aktiviert ist, erscheinen alle geänderten oder hinzugefügten Elemente eines Diagramms automatisch im "modifizierten" Stil. Wenn ein Element mit den Befehlen "Löschen" oder "Ausschneiden" gelöscht wird, wird es nicht aus dem Diagramm entfernt, sondern als "gelöscht" angezeigt. Auf diese Weise kann der Benutzer alle Änderungen eines Diagramms automatisch verfolgen.

Benutzen Sie den Befehl "**Bearbeiten/Stil/Alle gelöschten Elemente entfernen**", um die im "gelöschten" Stil angezeigten Elemente auch wirklich aus dem KOP/FBS-Diagramm zu entfernen. Dieser Befehl wird unabhängig von der aktuellen Auswahl ausgeführt und bezieht sich auf das gesamte Diagramm.

Um ein als "**gelöscht**" angezeigtes Element wiederherzustellen, wählt man das betreffende Element aus und ordnet es dem Stil "**Normal**", "**Modifiziert**" oder einem beliebigen "**Benutzer**"-Stil zu. Solche Operationen können zu ungültigen Verbindungen führen (mehr als eine Verbindung am gleichen Eingangspunkt angeschlossen), die bei der nächsten Programmprüfung erkannt werden.

A.8. Texteditor benutzen

Dieses Kapitel erläutert die speziellen Funktionalitäten und Befehle des ISaGRAF Texteditors, insbesondere für die Eingabe des Quellcodes von ST- und AWL-Programmen.

A.8.1. Bearbeitungsbefehle

Die Befehle im Menü "**Bearbeiten**" dienen der Textbearbeitung. Die meisten dieser Befehle beziehen sich auf die im Diagramm ausgewählten Textzeichen oder führen eine Aktion am aktuellen Standort des Eingabecursors aus.

Ausschneiden und Einfügen

Ausgewählter Text kann mit der ENTF-Taste gelöscht werden. Benutzen Sie den Befehl "**Bearbeiten / Widerrufen**", um Elemente nach einem ENTF-Befehl wiederherzustellen. Die Befehle "**Ausschneiden**", "**Kopieren**" und "**Einfügen**" im Menü "**Bearbeiten**" werden benutzt, um Texte im Programm zu verschieben oder zu kopieren oder um einen Textabschnitt, der aus einer anderen Anwendung stammt, aus der Zwischenablage einzufügen.

Suchen und Ersetzen

Die Befehle "**Bearbeiten / Suchen**" und "**Bearbeiten / Ersetzen**" werden benutzt, um Texte im Programm zu suchen und zu ersetzen. Beliebige Zeichenketten können gesucht werden. Die Suche kann vorwärts oder rückwärts verlaufen und fängt am aktuellen Standort des Eingabecursors an. Die Suche endet, entsprechend der gewählten Richtung, am Ende oder am Anfang des Programms.

Zu Zeile gehen

Der Befehl "**Bearbeiten / Zu Zeile gehen**" wird benutzt, um den Eingabecursor auf eine bestimmte Zeilennummer zu verlagern. Dieser Befehl ist nützlich, wenn man auf die nummerierte Zeile eines ST- oder AWL-Programms zugreifen möchte, in der der ISaGRAF Compiler einen Fehler erkannt hat.

Symbol aus dem Datenverzeichnis einfügen

Benutzen Sie den Befehl "**Bearbeiten / Variable einfügen**", um das Symbol einer im Datenverzeichnis des Projekts deklarierten Variablen oder eines Objekts am aktuellen Standort des Eingabecursors einzufügen. Das Symbol wird im Dialogfeld für die Variablenauswahl ausgewählt, welches im Kapitel "Mehr über die Programmeditoren" in diesem Dokument beschrieben wird.

Datei einfügen

Der Befehl "**Bearbeiten / Datei einfügen**" fügt den gesamten Inhalt einer Datei am aktuellen Standort des Eingabecursors ein. Beachten Sie, daß nur reine ASCII-Textdateien mit diesem Befehl bearbeitet werden können.

A.8.2. Optionen

Die Befehle im Menü "**Optionen**" werden benutzt, um die Editor-Befehlsleisten anzuzeigen oder nicht anzuzeigen und die Schriftart auszuwählen. Die ausgewählte Schriftart wird für die Textbearbeitung in der gesamten ISaGRAF Workstation benutzt.
Der Befehl "**Optionen / Schlüsselwörter anzeigen**" wird bei der Eingabe des Quellcodes eines ST- oder AWL-Programms benutzt, um die Toolbox mit den geläufigsten Schlüsselwörtern der ST- und AWL-Sprache anzuzeigen oder zu verstecken. Klicken Sie auf eine Schaltfläche in der Toolbox, um das entsprechende Schlüsselwort oder den entsprechenden Operatoren am aktuellen Standort des Eingabecursors einzufügen.

A.9. Mehr über die Programmeditoren

Dieses Kapitel enthält nützliche Informationen über die gemeinsamen Bearbeitungsfunktionen der ISaGRAF Programmeditoren. Sie betreffen hauptsächlich die Verknüpfungen mit anderen ISaGRAF Werkzeugen und die gemeinsamen ISaGRAF Dialogfelder.

A.9.1. Andere ISaGRAF Werkzeuge aufrufen

Programm prüfen (kompilieren)

Der Befehl "**Datei / Prüfen**" startet den ISaGRAF Code-Generator, um die Programmsyntax des zur Zeit bearbeiteten Programms zu prüfen. Bei der AS-Sprache werden die Ebenen 1 und 2 geprüft. Nach beendeter Syntaxprüfung muß das Fenster des Code-Generators geschlossen werden, um mit der Programmbearbeitung fortzufahren. Wenn die Anwendung nur ein Programm enthält (das bearbeitete Programm), wird der Anwendungscode generiert, wenn keine Syntaxfehler erkannt wurden. Der Befehl "**Optionen / Compiler-Optionen**" wird benutzt, um die Kompilierungs- und Optimierungsparameter einzustellen. Siehe Kapitel "Code-Generator benutzen" in diesem Handbuch für ausführliche Informationen über die Kompilierung und die Code-Entwicklung.

Anwendung simulieren oder debuggen

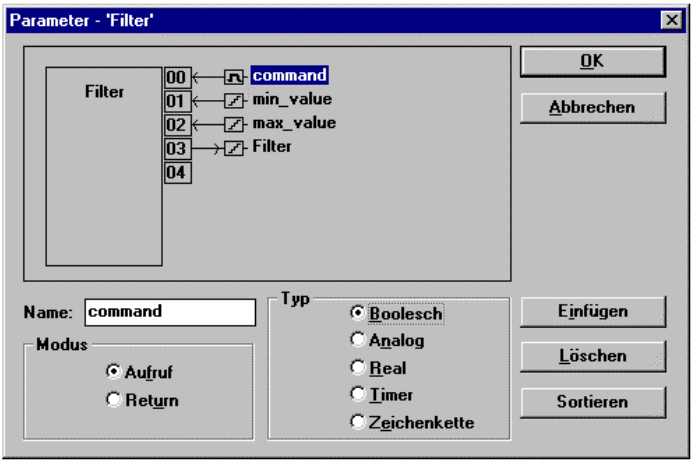
Die Befehle "**Datei / Simulieren**" und "**Datei / Debuggen**" starten den graphischen ISaGRAF Debugger, entweder im Simulationsmodus oder im Onlinemodus, und öffnen das bearbeitete AS-Programm wieder im Debugging-Modus. Im Debugging-Modus können keine Programmänderungen eingegeben werden.

Datenverzeichnis der Variablen bearbeiten

Der Befehl "**Datei / Datenverzeichnis**" wird benutzt, um das Datenverzeichnis der Variablen für die aktuelle Anwendung und das aktuelle Programm zu bearbeiten. Über diesen Befehl kann auch auf die Bearbeitung der Anwenderdefinitionen zugegriffen werden. Die **lokalen** Deklarationen oder Anwenderdefinitionen beziehen sich auf das momentan bearbeitete Programm.

A.9.2. Programmparameter

Wenn das bearbeitete Programm eine Funktion, ein Funktionsbaustein oder ein Unterprogramm ist, wird der Befehl "**Datei / Parameter**" benutzt, um ihre (seine) Aufruf- und Returnparameter zu definieren. Dieser Befehl ist wirkungslos, wenn das bearbeitete Programm ein AS- oder Hauptprogramm der Abschnitte **Anfang** oder **Ende** ist.
Unterprogramme, Funktionen und Funktionsbausteine können bis zu **32** Eingangs- oder Ausgangsparameter besitzen. Funktionen und Unterprogramme haben stets einen (und nur einen) Returnparameter, der den gleichen Namen wie die Funktion haben muß, gemäß der ST-Schreibkonventionen. Das folgende Dialogfeld wird benutzt, um die Parameter eines Unterprogramms zu beschreiben:



Die Liste links oben im Fenster zeigt die Parameter in der Reihenfolge des Aufrufmodells an: zuerst die Aufrufparameter, zuletzt die Returnparameter. Unterhalb dieser Liste erscheint eine detaillierte Beschreibung des in der Liste ausgewählten Parameters. Für die Parameter können alle ISaGRAF Datentypen benutzt werden. Die Returnparameter müssen nach den Aufrufparametern in der Liste erscheinen. Bei der Parameterbenennung müssen die folgenden Regeln beachtet werden:

- Der Name darf nicht länger als **16** Zeichen sein.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Die weiteren Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Der Befehl "**Einfügen**" wird benutzt, um einen neuen Parameter vor dem ausgewählten Parameter einzufügen. Der Befehl "**Löschen**" wird benutzt, um den ausgewählten Parameter zu löschen. Der Befehl "**Sortieren**" sortiert die Parameter automatisch, so daß die Returnparameter an das Ende der Liste gesetzt werden.

A.9.3. Andere Befehle des "Datei"-Menüs

Die folgenden Befehle sind im "**Datei**"-Menü aller Programmeditoren verfügbar:

Ein anderes Programm öffnen

Mit dem Befehl "**Datei / Öffnen**" kann das in Arbeit befindliche Programm geschlossen und die Bearbeitung eines anderen Programms des aktuellen Projekts in der gleichen Sprache aufgenommen werden. Diese Funktion kann nicht benutzt werden, um ein Programm zu bearbeiten, das in einer anderen Sprache geschrieben wurde. Das neue Programm ersetzt das aktuelle Programm im Bearbeitungsfenster.

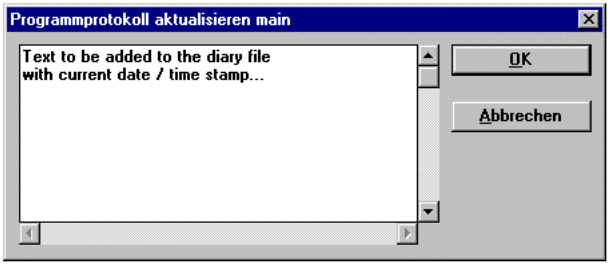
Programm drucken

Der Befehl "**Datei / Drucken**" sendet das bearbeitete Programm zum Drucker. Der ISaGRAF Dokumentationsgenerator startet automatisch, um das bearbeitete Programm und die zugeordneten lokalen Variablen auszudrucken.
Bei manchen grafischen Programmen (AS, FBS und Schneller KOP) kann eine Abbildung des betreffenden Netzwerks mit dem Befehl "**Bearbeiten / Zeichnung kopieren**" im Metafile-Format in die Zwischenablage kopiert werden. Die Abbildung kann daraufhin in andere Anwendungen, z.B. Textverarbeitungsprogramme, eingefügt werden. Bei AS-Programmen erscheinen lediglich die Daten der Ebene 1 (Netzwerk, Nummerierung und Kommentare der Ebene 1) in der kopierten Metafile-Datei.

A.9.4. Programmprotokoll aktualisieren

Der Befehl "**Datei / Programmprotokoll**" öffnet die mit einem Programm verknüpfte Protokolldatei, in die der Anwender Daten bezüglich der Programmentwicklung eingeben kann. Außerdem wird die Protokolldatei bei jeder Programmkompilierung automatisch mit Syntaxprüfmeldungen, mit Datum und Uhrzeit der Kompilierung, aktualisiert.

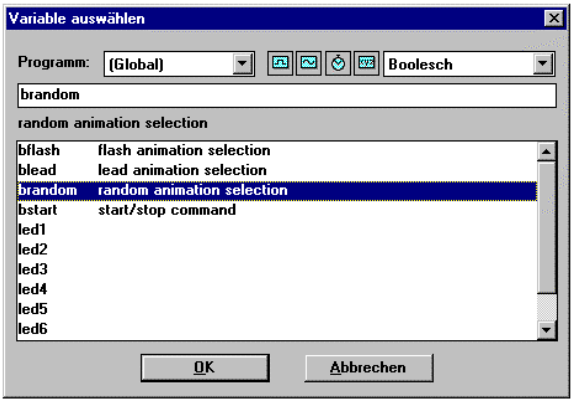
Wenn der Modus "**Programmprotokoll aktualisieren**" im Menü "**Optionen**" eines Programm-Editors aktiviert wird, öffnet sich das folgende Dialogfeld jedesmal dann, wenn das Programm auf der Festplatte gespeichert wird.



Wenn Sie OK klicken, wird der eingegebene Text am Ende der Programmprotokolldatei mit laufendem Datum und Uhrzeit gespeichert. Die Protokolldatei ist besonders bei der Instandhaltung eines Programms von Nutzen, da sie über seinen Lebenszyklus Auskunft gibt.

A.9.5. Eine Variable aus dem Datenverzeichnis auswählen

Bei der Bearbeitung eines Textprogramms (ST oder AWL) ermöglicht der Befehl "**Bearbeiten / Variable einfügen**", den Namen einer deklarierten Variablen auszuwählen und ihn am aktuellen Standort des Eingabecursors einzufügen. Bei der Bearbeitung von KOP- oder FBS-Programmen ist die Variablenauswahl für die Beschreibung der Kontakte und Spulen, der E/A-Parameter von Funktionsbausteinen und der FBS-Variablenrechtecken erforderlich. In beiden Fällen öffnet sich das folgende Dialogfeld, in dem eine deklarierte Variable ausgewählt werden kann:



Das Textfeld "**Geltungsbereich**" wird benutzt, um zwischen globalen und lokalen Variablen zu wählen. Das Textfeld rechts im Bildschirm ermöglicht die Auswahl des Datentyps. Die kleinen Symbole links neben dem Textfeld sind Schaltflächen, die für die Auswahl der geläufigsten Datentypen benutzt werden können:

 ..... Boolesch

 ..... Ganzzahlig / Real

 ..... Timer

 ..... Zeichenkette

Um eine Variable auszuwählen, klicken Sie auf ihren Namen in der Liste. Ihr Name und ihr Kommentar erscheinen nun über der Liste im Textfeld. Klicken Sie "**OK**", um die Auswahl zu bestätigen. Sie können auch einen Variablennamen direkt in das Textfeld eingeben, ohne die Liste zu benutzen.

A.9.6. Das Ausgabefenster

Die folgenden Befehle sind im Werkzeugmenü aller Spracheditoren verfügbar. Sie werden benutzt, um Informationen in einem kleinen Textfeld unten im Bearbeitungsfenster anzuzeigen, die benutzt werden können, um auf entsprechende Daten in den Programmen zuzugreifen.

"Compilerausgaben anzeigen": Anzeige der Fehlermeldungen der letzten Kompilierung des in Arbeit befindlichen Programms im Ausgabefenster.
"Suchen in…": Textvorkommen im bearbeiteten Programm suchen und im Ausgabefenster auflisten. Bei den AS- und FD-Sprachen sucht dieser Befehl in allen Programmen der Ebene 2.
"Ausgabefenster verstecken": schließt das Ausgabefenster.

Wenn Fehlermeldungen und Textvorkommen im Ausgabefenster angezeigt werden, kann die Auswahl durch Doppelklicken auf eine Zeile unmittelbar auf die

entsprechende Stelle im Programm verlagert werden. Bei AS- und FD-Programmen öffnet diese Aktion das entsprechende Programmierfenster der Ebene 2.

A.10. Datenverzeichnis-Editor benutzen

Der ISaGRAF Datenverzeichnis-Editor ist ein Bearbeitungswerkzeug für die Deklarationen der internen und E/A-Variablen, der Funktionsbaustein-Instancen und der "Definitionen" einer Anwendung. Im Datenverzeichnis sind die deklarierten Variablen und Funktionsbaustein-Instancen der Anwendung, sowie die Definitionen (konstanten Zeichenfolgen) zusammengefaßt.
Alle Variablen, Funktionsbausteine und Definitionen müssen zuerst im Datenverzeichnis deklariert werden, bevor sie im Quellcode benutzt werden können. Variablen und Definitionen können mit allen Automatisierungssprachen (AS, FBS, KOP, ST und AWL) benutzt werden. Funktionsbausteine, die in der FBS-Sprache benutzt werden, brauchen nicht deklariert zu werden, da die ISaGRAF Editoren der FBS und des Schnellen KOP die Instancen der benutzten Bausteine automatisch deklarieren.

Variablen

Die Variablen werden nach **Geltungsbereich** und **Typ** sortiert. Ausschließlich Variablen eines gleichen Typs und Geltungsbereichs können zusammen in einem Eingaberaster eingegeben werden. Grundlegende Geltungsbereiche für Variablen sind:

GLOBAL kann von allen Programmen des aktuellen Projekts benutzt werden.

LOKAL kann nur von einem Programm benutzt werden.

Grundlegende Variablentypen sind:

BOOLESCH true/false binäre Werte

ANALOG reale oder ganzzahlige Werte

TIMER Zeitwerte

ZEICHENKETTE Zeichenfolgen

Variablen werden mit einem Namen, einem Kommentar, einem Attribut, einer Netzwerkadresse und anderen spezifischen Feldern gekennzeichnet. Grundlegende Variablenattribute sind:
INTERN..... Speichervariable
EINGANG ..... mit einer Eingangseinrichtung verknüpfte Variable
AUSGANG..... mit einer Ausgangseinrichtung verknüpfte Variable
KONSTANT ..... interne nur-lesen Variable (mit Initialwert)

Achtung: **Timer** sind immer **interne** Variablen. **Eingangs-** und **Ausgangs**variablen haben immer einen **GLOBALEN** Geltungsbereich.

Definitionen

Eine Definition ist ein Alias, der in einer beliebigen Sprache benutzt werden kann, um eine Zeichenfolge zu ersetzen. Der ersetzte Text kann ein Variablenname, ein konstanter Ausdruck oder ein komplexer Ausdruck sein. Definitionen werden gemäß ihres Geltungsbereichs sortiert. Ausschließlich Definitionen mit gleichem Typ und

Geltungsbereich können zusammen in einem Eingaberaster eingegeben werden. Grundlegende Geltungsbereiche sind:

ALLGEMEIN	kann von allen Programmen aller Projekte benutzt werden
GLOBAL	kann von allen Programmen des aktuellen Projekts benutzt werden
LOKAL	kann nur von einem Programm benutzt werden

Definitionen werden mit einem Namen, einer genau definierten ST-Textäquivalenz und einem freien Kommentar gekennzeichnet.

Funktionsbaustein-Instancen

Die Instancen der in den ST- und AWL-Sprachen benutzten Funktionsbausteine müssen im Datenverzeichnis deklariert werden. Da Funktionsbausteine "versteckte" interne Daten besitzen, muß jede Kopie eines Funktionsbausteins identifiziert werden. Das folgende Beispiel zeigt den in der Bibliothek definierten Funktionsbaustein "R_TRIG" (Erkennung positiver Flanke), der für die Flankenerkennung unterschiedlicher Variablen benutzt wird. Jede Kopie des Baustein muß mit einem einmaligen Namen gekennzeichnet werden. Die Benennung des Bausteins und die Definition seiner Parameter erfolgt mit dem Bibliotheksmanager:

Bausteinname:	**R_TRIG**
***Parameter*:**	**Eingang=CLK**
	Ausgang=Q

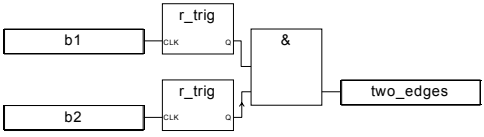
Die Benennung der Instancen erfolgt mit dem Datenverzeichnis-Editor:

***Instancenname*:**	**TRIG_B1**	***Bausteinname*:**
	R_TRIG	
***Instancenname*:**	**TRIG_B2**	***Bausteinname*:**
	R_TRIG	

Die deklarierten Instancen können in ST-Programmen benutzt werden:

```
TRIG_B1 (b1);
edge_b1 := TRIG_B1.Q;      (* b1 Variablenflanken-Erkennung *)
TRIG_B2 (b2);
edge_b2 := TRIG_B2.Q;      (* b2 Variablenflanken-Erkennung *)
```

Deklarierte Funktionsbaustein-Instancen können **GLOBAL** (allen Programmen des Projekts bekannt) oder programm-**LOKAL** sein. Funktionsbausteine, die in der FBS- oder KOP-Sprache benutzt werden, brauchen nicht deklariert zu werden, da der ISaGRAF FBS-Editor die Instancen der benutzten Bausteine automatisch deklariert.



(* Funktionbausteine tragen immer den Namen des in der Bibliothek definierten Bausteins. Die ISaGRAF Editoren der FBS und des Schnellen KOP deklarieren jedesmal eine Instance, wenn ein Baustein in das Diagramm eingefügt wird *)

Die automatisch von den FBS- und Schnellen KOP-Editoren deklarierten Funktionsbaustein-Instancen sind immer LOKAL zum bearbeiteten Programm.

Netzwerkadressen

Netzwerkadressen sind **optionell**. Eine Variable mit einer Netzwerkadresse ungleich null kann bei der Ausführung von einem externen System **überwacht** werden (beispielsweise von einem Prozeßvisualisierungssystem). Im allgemeinen liefert die Netzwerkadresse einen Identifizierungsmechanismus für Kommunikationssysteme, die keine symbolischen Namen verarbeiten können. Die Netzwerkadresse wird bei der Erstellung oder Änderung einer Variablen eingegeben (der Variablenbeschreibung hinzugefügt.)

A.10.1. Das Fenster des Datenverzeichnisses

Das Datenverzeichnis-Bearbeitungsfenster zeigt eine Liste von Variablen mit gleichem Typ und Geltungsbereich an. Typ und Geltungsbereich der bearbeiteten Variablen erscheinen in der Titelzeile.

Im Bearbeitungsfenster erscheinen lediglich die hauptsächlichen Felder der Variablenbeschreibung: Name, Attribut und Netzwerkadresse, sowie Kommentartext. Die vollständige Beschreibung der ausgewählten Variablen erscheint in der Statuszeile.

Benutzen Sie die folgenden Schaltflächen in der Befehlsleiste, um den Geltungsbereich der zu bearbeitenden Variablen auszuwählen:

ALLGEMEIN kann von allen Programmen aller Projekte benutzt werden

GLOBAL kann von allen Programmen des aktuellen Projekts benutzt werden

LOKAL kann nur von einem Programm benutzt werden

Benutzen Sie die in der Titelzeile angezeigten "Tab"-Felder, um den Typ des zu bearbeitenden Objekts auszuwählen:

Benutzen Sie das Textfeld auf der linken Seite der Befehlsleiste, um das Namensvorzeichen einer Variablen zu suchen. In diesem Fall erstreckt sich die Suche über die gesamte Liste und fängt bei der aktuellen Auswahl an. Der Befehl "**Bearbeiten** / **Suchen**" kann auch für die Suche einer Zeichenfolge in Variablennamen oder Kommentaren benutzt werden. Die Auswahl wird auf die gefundene Variable verlagert. Zwischen **Groß**- und **Kleinschreibung** wird immer **unterschieden**.

A.10.2. Variablen verwalten

Die Befehle im "**Datei**"-Menü verarbeiten eine ausgewählte Klasse von Variablen, Funktionsbaustein-Instancen oder Definitionen. Benutzen Sie den Befehl "**Anderer Typ**", um den Typ und Geltungsbereich der zu bearbeitenden Objekte auszuwählen.

Variablen drucken

Benutzen Sie den Befehl "**Datei / Drucken**", um die in Arbeit befindliche Liste (Variablen oder Definitionen) mit Hilfe des ISaGRAF Dokumentationsgenerators auf einem Windows™ Standarddrucker auszudrucken. Der Ausdruck beinhaltet eine vollständige Beschreibung jeder Variablen oder Definition des in Arbeit befindlichen Typs.

Neue Variablen erstellen

Der Befehl "**Bearbeiten / Neu**" wird benutzt, um neue Variablen, Funktionsbaustein-Instancen oder Definitionen für den ausgewählten Geltungsbereich und Typ zu erstellen. Die neuen Variablen werden vor der ausgewählten Variablen eingefügt. Wenn dieser Befehl eingeleitet wird, öffnet sich ein Dialogfeld für die Eingabe der Variablenbeschreibung. Wenn die Beschreibung abgeschlossen ist, klicken Sie auf die Schaltfläche "**Sichern**", um die Variable in die Liste einzufügen. Das Dialogfeld öffnet sich wieder, so daß der Benutzer weitere Variablen eingeben kann, ohne den Befehl "**Bearbeiten**" wiederholen zu müssen. Klicken Sie auf "**Abbrechen**", um die Variablenerstellung abzubrechen.

Bestehende Variablen modifizieren

Der Befehl "**Bearbeiten**" im Menü "**Bearbeiten**" kann benutzt werden, um die Beschreibung der ausgewählten Variablen zu modifizieren. Wenn dieser Befehl eingeleitet wird, öffnet sich ein Dialogfeld, in dem die Variablenbeschreibung geändert werden kann. Wenn die Beschreibung abgeschlossen ist, klicken Sie auf die Schaltfläche "**Sichern**", um die Änderung vorzunehmen. Mit den Schaltflächen "**Nächste**" und "**Vorherige**" kann der Änderungsbefehl auf Nachbarvariablen erweitert werden. Klicken Sie auf "**Abbrechen**", um das Dialogfeld zu schließen, ohne daß die Änderungen gespeichert werden.

Ausschneiden und Einfügen

Der ISaGRAF Datenverzeichnis-Editor ermöglicht die **Auswahl mehrerer Zeilen**. Verschiedene Befehle sind verfügbar, um die aktuelle Variablenliste zu bearbeiten. Folgende Befehle sind im Menü "**Bearbeiten**" verfügbar:

KOPIEREN ..... Die ausgewählte Variablengruppe in die Datenverzeichnis-Zwischenablage kopieren
AUSSCHNEIDEN.. Die ausgewählte Variablengruppe in die Ablage kopieren und aus der aktuellen Liste entfernen
LÖSCHEN..... Die ausgewählte Variablengruppe aus der aktuellen Liste entfernen
EINFÜGEN..... Den Inhalt der Datenverzeichnis-Zwischenablage vor der ausgewählten Variablen einfügen

Die Kopier-, Ausschneide- und Einfügefunktionen können zwischen mehreren Variablenlisten benutzt werden, jedoch nicht zwischen Listen verschiedener Objekttypen.

Variablen sortieren

Der Befehl "**Werkzeuge / Sortieren**" sortiert die Variablen oder Definitionen der aktuellen Liste. Die Variablen werden entsprechend ihrer Attribute sortiert:

- zuerst die internen Variablen
- dann die Eingangsvariablen
- zuletzt die Ausgangsvariablen

Variablen mit gleichen Attributen werden in alphabetischer Reihenfolge sortiert. Definitionen werden ebenfalls in alphabetischer Reihenfolge sortiert.

Netzwerkadressen bestimmen

Netzwerkadressen sind **optionell**. Eine Variable mit einer Netzwerkadresse ungleich null kann bei der Ausführung von einem externen System **überwacht** werden (beispielsweise von einem Prozeßvisualisierungssystem). Die Netzwerkadresse wird der Beschreibung einer Variablen bei deren Erstellung oder Änderung beigefügt. Der Befehl "**Werkzeuge / Adressen neu nummerieren**" wird benutzt, um die Netzwerkadressen einer ganzen Variablengruppe zu setzen. Wenn dieser Befehl eingeleitet wird, verarbeitet er die in der Liste ausgewählte Variablengruppe. Wenn eine hexadezimale **Basisadresse** eingegeben wird (Adresse für die erste Variable in der Gruppe), werden die Netzwerkadressen der Variablen in der Gruppe mit **fortlaufenden Adressen** erstellt. Wenn ein Basisadresse gleich null eingeben wird, werden die Netzwerkadressen sämtlicher ausgewählten Variablen auf null zurückgesetzt.

Boolesche "true/false" Zeichenfolgen importieren

Bei der Bearbeitung von Definitionen wird der Befehl "**Werkzeuge / True/False-Definitionen importieren**" benutzt, um Zeichenfolgen, die mit Booleschen Variablen verbunden sind, um die Zustände TRUE und FALSE darzustellen, automatisch als Sprach-Schlüsselwörter zu definieren. Solche Zeichenfolgen werden normalerweise für die Debugging-Formatierung definiert. Sie müssen als Definitionen spezifiziert werden, wenn sie in den Programmen benutzt werden sollen. Dieser Befehl sucht Boolesche true/false Zeichenfolgen (mit dem gleichen Geltungsbereich wie die ausgewählte Zeichenfolge) in den Deklarationen der Definitionen.

A.10.3. Objektbeschreibung

Für jede Variable, Funktionsbaustein-Instance und Definition muß eine genaue Beschreibung eingegeben werden. Die Beschreibungsfelder sind bei jedem Objekttyp unterschiedlich. Die folgenden Felder werden für alle Variablentypen benutzt:

Name ..... Name der Variablen: Das erste Zeichen muß ein Buchstabe sein, weitere Zeichen müssen Buchstaben, Zahlen oder '_' sein.

Netzwerkadresse .....Hexadezimale Netzwerkadresse (optionell). Wenn dieses Feld ungleich null ist, kann die Variable von externen Systemen bei der Ausführung überwacht werden.
Kommentar .....Freier Kommentar für die Variablenbeschreibung.
Nicht-flüchtig .....Diese Option bestimmt, daß die Variable im Sicherungsspeicher gespeichert werden muß.

Diese Beschreibungsfelder werden für **Boolesche** Variablen benutzt:
Attribut .....Spezifiziert eine interne, konstante, Ein- oder Ausgangsvariable.
"False"-Zeichenfolge.....Zeichenfolge, die beim Debugging für den False-Wert benutzt wird.
"True"-Zeichenfolge .....Zeichenfolge, die beim Debugging für den True-Wert benutzt wird.
Bei Init auf true setzen ..Der Initialwert ist TRUE, wenn diese Option aktiviert wird, sonst ist der Initialwert FALSE.

Diese Beschreibungsfelder werden für **ganzzahlige oder reale** Variablen benutzt:
Attribut .....Spezifiziert eine interne, konstante, Ein- oder Ausgangsvariable.
Format.....Spezifiziert eine ganzzahlige oder reale (Gleitpunkt-) Variable. Das Anzeigeformat für den Debugging-Vorgang kann ausgewählt werden.
Einheit-Zeichenfolge .....Zeichenfolge, die benutzt wird, um die physikalische Einheit beim Debugging zu identifizieren.
Umrechnung .....Name der mit einer Variablen verbundenen Umrechnungstabelle oder -funktion (nur für Ein- oder Ausgangsvariablen)
Initialwert .....Initialwert der Variablen (muß das gleiche Format wie die Variable haben). Wenn nicht spezifiziert, ist der Initialwert 0.

Diese Beschreibungfelder werden für **Timer**-Variablen benutzt:
Attribut .....Spezifiziert eine interne oder konstante Variable.
Initialwert .....Initialwert der Variablen (Zeitwert). Wenn nicht spezifiziert, ist der Initialwert time#0s.

Diese Beschreibungsfelder werden für **Zeichenketten**-Variablen benutzt:
Attribut .....Spezifiziert eine interne, konstante, Ein- oder Ausgangsvariable.
Maximale Länge.....Spezifiziert die Anzahl an Zeichen, die maximal in der Zeichenkette gespeichert werden können.
Initialwert .....Initialwert der Variablen (die Länge kann die Kapazität der Zeichenkette nicht überschreiten). Wenn nicht spezifiziert, ist der Initialwert eine leere Zeichenfolge.

Diese Beschreibungsfelder werden für **Definitionen** benutzt:
Name .....Name, der in den ST-Quelldateien benutzt wird: Das erste Zeichen muß ein Buchstabe sein, weitere Zeichen müssen Buchstaben, Zahlen oder '_' sein.

Äquivalenz .....Zeichenfolge gemäß der ST-Syntax, die die Definition beim Kompilieren ersetzt. Beispiel: Name = PI - Äquivalenz = 3.14159
Kommentar .....Freier Kommentar für die definierte Äquivalenz

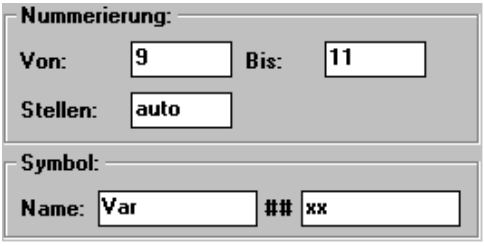
Diese Beschreibungsfelder werden für **Funktionsbaustein-Instancen** benutzt:
Name .....Name der Instance, der in den ST-Quelldateien benutzt wird: Das erste Zeichen muß ein Buchstabe sein, weitere Zeichen müssen Buchstaben, Zahlen oder '_' sein.
Typ.....Name des entsprechenden Funktionsbausteins in der Bibliothek.
Kommentar .....Freier Kommentar für die Funktionsbaustein-Instance.

A.10.4. Schnelle Deklaration

Mit dem Befehl "**Werkzeuge / Schnelle Deklaration**" können mehrere Variablen gleichzeitig deklariert werden. Diese Variablen werden gemäß einer Nummerierungskonvention benannt. Folgende Definitionen sind erforderlich:
- Index (Nummer) der ersten und der letzten Variablen,
- der Text, der vor und hinter der Nummer im Variablensymbol erschienen soll,
- die Anzahl der Stellen, die die Nummer im Variablensymbol haben soll.

Zusätzlich können die Attribute der erstellten Variablen (intern, Eingang oder Ausgang...) sowie einige Eigenschaften, die vom Variablentyp (Attribut nicht-flüchtig, ganzzahliges oder reales Format, maximale Länge einer Zeichenkette) abhängig sind, spezifiziert werden.
Der vor der Nummer der Variablen einzufügende Text muß definiert werden, da ein Variablensymbol nicht mit einer Zahl beginnen kann. Wenn "Stellenzahl" auf "Auto" gesetzt wird, formatiert ISaGRAF die Zahl der Stellen auf die geringste erforderliche Stellenzahl. Wenn die Stellenzahl definiert wird, formatiert ISaGRAF alle Nummern mit vorgesetzten "0"-Zeichen auf die spezifizierte Länge. Wenn die Stellenzahl von vornherein festgelegt wird, kann eine falsche lexikalische Anrdnung vermieden werden. Hier einige Beispiele.

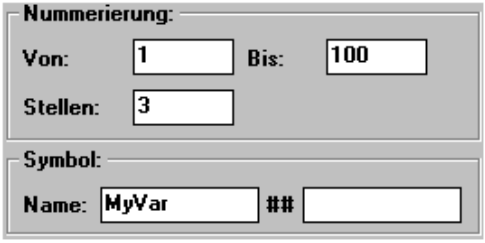
Beispiel Diese Einstellung für eine schnelle Variablendeklaration:



erstellt die drei folgenden Variablen:

Var9xx Var10xx Var11xx

Beispiel Diese Einstellung für eine schnelle Variablendeklaration:



erstellt 100 Variablen mit Namen von MyVar001 bis MyVar100

A.10.5. Modbus SCADA Adressierbereich

ISaGRAF "Netzwerkadressen" werden häufig benutzt, um eine Verbindung zwischen dem ISaGRAF System und einem SCADA herzustellen (Modbus Kommunikation). In diesem Fall ist das SCADA der Modbus Master und das ISaGRAF Zielsystem verhält sich wie ein Modbus Slave. Die Netzwerkadressen werden benutzt, um einen virtuellen Modbus-Bereich (Map) für alle ISaGRAF Variablen zu erstellen, die vom SCADA gesteuert werden. Der Befehl "**Werkzeuge / Modbus SCADA Adressierbereich**" leitet ein leistungsfähiges Werkzeug ein, um auf schnellem Wege einen virtuellen Bereich mit den Variablen der Anwendung zu erstellen.
Die Mapping-Werkzeuge zeigen zwei Listen. Die obere Liste ist ein Segment (4096 Plätze) des Modbus-Bereichs, in dem die abgebildeten Variablen (die Netzwerkadressen besitzen) erscheinen. Die untere Liste zeigt unabgebildete Variablen (deren Netzwerkadressen nicht definiert sind). Die Adresse "0" kann nicht benutzt werden, um eine Variable abzubilden.
Benutzen Sie die Befehle "**Abbilden**" und "**Entfernen**" im Menü "**Bearbeiten**" um Variablen von einer Liste in die andere zu verlagern und auf diese Weise den Bereich (Map) aufzubauen. Diese Aktionen können auch durch Doppelklicken auf Variablen in der Liste ausgeführt werden. Es ist jederzeit möglich, über das Listenfeld "Segment" auf ein anderes Segment der Liste zuzugreifen.
Die Befehle des Menüs "**Optionen**" werden benutzt, um die Adressen entweder in dezimal oder in hexadezimal anzuzeigen.
Der Befehl "**Bearbeiten / Suchen**" wird für die Suche einer deklariereten Variablen benutzt, ob bereits abgebildet oder nicht.

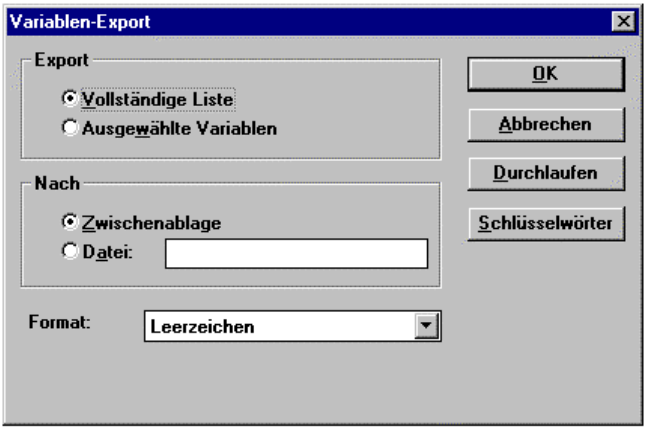
A.10.6. Datenaustausch mit anderen Anwendungen

Der ISaGRAF Datenverzeichnis-Editor besitzt Import- und Exportfunktionen, mit deren Hilfe Daten mit anderen Anwendungen ausgetauscht werden können, z.B. mit Textverarbeitungsprogrammen, Tabellenrechnungsprogrammen, Datenbank-Managern... Diese Befehle sind im Menü "**Bearbeiten**" zusammengefaßt. Der Befehl "**Text exportieren**" generiert eine reine ASCII-Textbeschreibung der Beschreibungsfelder einer Objektgruppe und speichert diesen Text entweder in der Windows-Zwischenablage oder in einer Datei. Diese Daten sind dazu bestimmt, von anderen Anwendungen übernommen zu werden. Der Befehl "**Text importieren**" importiert die in einem ASCII-Textformat geschriebenen Deklarationsfelder von

Variablen, entweder aus der Windows-Zwischenablage oder aus einer Datei, und aktualisiert die momentan bearbeitete Liste mit den importierten Feldern. Solche Daten stammen immer aus einer anderen Anwendung.

Daten exportieren

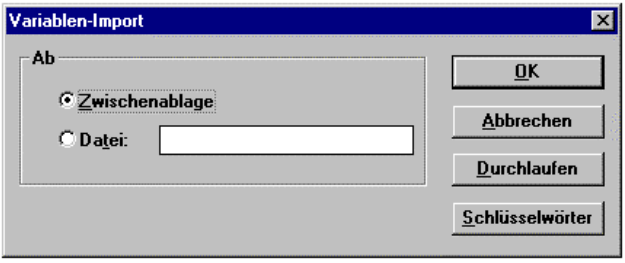
Wenn der Befehl "**Text exportieren**" gestartet wird, erscheint das folgende Dialogfeld, in dem der Anwender den Exportvorgang steuern kann.



Die Auswahl "**Vollständige Liste**" bedeutet, daß die gesamte bearbeitete Liste exportiert wird. In diesem Fall wird die aktuelle Auswahl ignoriert. Die Auswahl "**Ausgewählte Variablen**" bedeutet, daß nur die ausgewählten Variablen exportiert werden.
Wenn die Option "**Zwischenablage**" aktiviert wird, dann werden die exportierten Daten in der Windows-Zwischenablage in einem reinen ASCII-Textformat gespeichert. Der Text kann mit Hilfe der Befehle "Einfügen" in andere Anwendung eingefügt werden. Wenn die Option "**Datei**" aktiviert wird, wird der exportierte Text in einer ASCII-Datei gespeichert. Hierzu muß der vollständige Pfad für diese Datei eingegeben werden. Mit dem Befehl "**Durchlaufen**" können vorhandenen Pfadnamen gesucht werden.
Nun wählt der Anwender das Format des exportierten Texts. Die verfügbaren Formate werden in den folgenden Abschnitten beschrieben. Klicken auf die Schaltfläche "**OK**" startet die Exportfunktion. Klicken auf die Schaltfläche "**Abbrechen**" schließt das Dialogfeld, ohne die Funktion zu starten.
Alle Felder der ausgewählten Objekte erscheinen im exportierten Text in der Standard-Deklarationsordnung. Die erste Zeile des exportierten Texts enthält die Feldnamen. Jedes Objekt wird auf einer Textzeile beschrieben. Das Trennzeichen am Zeilenende ist die MS-DOS-Standardsequenz "**0d-0a**". Die Namen auf der ersten Textzeile, die für die Kennzeichnung der Felder benutzt werden, können durch Klicken auf die Schaltfläche "**Schlüsselwort**" geändert werden. Dieser Befehl wird in den folgenden Abschnitten beschrieben.

Daten importieren

Wenn der Befehl "**Text importieren**" gestartet wird, erscheint das folgende Dialogfeld, in dem der Anwender den Importvorgang steuern kann.



Wenn die Option "**Zwischenablage**" aktiviert wird, dann werden die importierten Daten aus der Windows-Zwischenablage in einem reinen ASCII-Textformat importiert. Wenn die Option "**Datei**" aktiviert wird, wird der importierte Text aus einer ASCII-Datei importiert. Hierzu muß der vollständige Pfad für diese Datei eingegeben werden. Mit dem Befehl "**Durchlaufen**" können vorhandenen Pfadnamen gesucht werden.
Die Importfunktion erkennt das für den importieretn Text benutzte Format (Trennzeichen) automatisch. Verfügbare Formate werden in den folgenden Abschnitte beschrieben. Klicken auf die Schaltfläche "**OK**" startet die Importfunktion. Klicken auf die Schaltfläche "**Abbrechen**" schließt das Dialogfeld, ohne die Funktion zu starten. Die Namen auf der ersten Textzeile, die für die Kennzeichnung der importierten Felder benutzt werden, können durch Klicken auf die Schaltfläche "**Schlüsselwort**" geändert werden. Dieser Befehl wird in den folgenden Abschnitten beschrieben.
Die erste Textzeile muß die Feldnamen enthalten, gemäß der in den folgenden Zeilen benutzten Reihenfolge. Jedes Objekt muß auf einer Textzeile beschrieben werden. Das Trennzeichen am Zeilenende ist die MS-DOS-Standardsequenz "**0d-0a**". Felder können in einer beliebigen Reihenfolge erscheinen. Wenn bestimmte Felder fehlen, werden sie automatisch im importierten Objekt durch Vorgabewerte ersetzt. Wenn ein importiertes Objekt bereits in der bearbeiteten Liste vorkommt, muß der Anwender eine Überschreiberlaubnis erteilen. Die Objektbeschreibung wird daraufhin mit den importierten Feldern aktualisiert. Wenn bestimmte Felder fehlen, werden sie nicht in der Objektbeschreibung aktualisiert.

Verfügbare Textformate

Hier die Liste der verfügbaren Formate für die Exportfunktion. Diese Formate werden automatisch von der Importfunktion erkannt.

- Tabulationen

Beschreibung: Die Felder sind durch Tabulationen getrennt.
Beispiel:

Name	Attribute	Comment
level	internal	internal calculated water level
alrm1	output	main alarm output

- Kommas

Beschreibung: Die Felder sind durch Kommas getrennt.
Beispiel: Name,Attribute,Comment
level,internal,internal calculated water level
alrm1,output,main alarm output

- Semikola

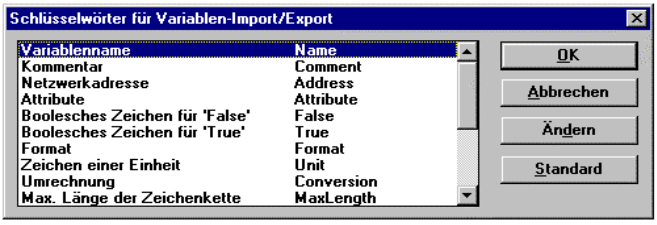
Beschreibung: Die Felder sind durch Semikola getrennt.
Beispiel: Name;Attribute;Comment
level;internal;internal calculated water level
alrm1;output;main alarm output

- Kommas und Anführungszeichen

Beschreibung: Die Felder sind durch Kommas getrennt.
Jedes Feld steht in Anführungszeichen.
Beispiel: "Name","Attribute","Comment"
"level","internal","internal calculated water level"
"alrm1","output","main alarm output"

Schlüsselwörter

Die Namen, die auf der ersten Textzeile als Kennzeichnung der importierten und exportierten Felder benutzt werden, können durch Klicken auf die Schaltfläche "**Schlüsselwort**" geändert werden. Dieser Befehl öffnet das folgende Dialogfeld:



In diesem Fenster sind sämtliche Objektfelder mit den ihnen zugeordneten Schlüsselwörtern aufgelistet. Um ein Schlüsselwort zu ändern, wählen Sie ein Feld in der Liste und klicken auf die Schaltfläche "**Ändern**". Die Schaltfläche "**Vorgabe**" stellt die ursprüngliche Schlüsselwörterliste wieder her. Bei der Benennung der Schlüsselwörter müssen die folgenden Regeln beachtet werden:

- Der Name darf nicht länger als **16** Zeichen sein.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Die weiteren Zeichen können **Buchstaben**, **Zahlen** oder '_' sein.
- Ein Name kann nicht für verschiedene Schlüsselwörter benutzt werden.

Hier die in ISaGRAF benutzten Standard-Schlüsselwörter:

Objektname.....**Name**
Textkommentar .....**Comment**
Netzwerkadresse.....**Address**
Attribute (intern, Ein/Ausgang).....**Attribute**

Boolesche 'False'-Zeichenkette .....**False**
Boolesche 'True'-Zeichenkette.....**True**
Analoges Format (real/ganzzahlig) .....**Format**
Analoge Einheit-Zeichenkette.....**Unit**
Name einer analogen Umrechnung .....**Conversion**
Maximale Zeichenkettenlänge .....**MaxLength**
Funktionsbaustein-Bibliothek (Typ).....**Library**
Äquivalenz einer Definition .....**Equivalence**
Attribut: Intern.....**Internal**
Attribut: Eingang.....**Input**
Attribut: Ausgang.....**Output**
Attribut: Konstant.....**Constant**
Reales analoges Format .....**Real**
Ganzzahliges analoges Format .....**Integer**

A.11. E/A-Verdrahtungseditor benutzen

Ziel der E/A-Verdrahtung ist es, eine logische Verbindung zwischen den E/A-Variablen einer Anwendung und den physikalischen Kanälen der Karten einer Zielsteuerung herzustellen. Um diese Operation auszuführen, müssen alle Karten der Zielsteuerung identifiziert und eingesetzt und die E/A-Variablen an die entsprechenden E/A-Kanäle angeschlossen werden.
In der linken Fensterhälfte erscheint eine Abbildung des Baugruppenträgers der Zielsteuerung mit seinen **Kartensteckplätzen**. Die Steckplätze können unbenutzt sein oder E/A-Karten oder komplexe Baugruppen enthalten. Jeder Steckplatz ist mit einer **Ordnungsnummer** gekennzeichnet. Der Baugruppenträger kann bis zu **255** Karten enthalten. Die Liste in der rechten Fensterhälfte zeigt die Parameter der ausgewählten Karte und die an diese Karte angeschlossenen Variablen an. Eine Karte kann bis zu **128** E/A-Kanäle besitzen. Die Gesamtzahl einzelner E/A-Karten (einschließlich einzelner Baugruppen und Karten komplexer Baugruppen) begrenzt sich auf **255**.

Symbole

Die Symbole auf der Vorderseite einer Karte kennzeichnen den Typ und die Attribute der Variablen, die an die Kartenkanäle angeschlossen werden können. Unter ISaGRAF können Variablen unterschiedlichen Typs nicht an dieselbe Karte angeschlossen werden. Folgende Symbole werden benutzt:

- ..... Typ: Boolesch
- ..... Typ: ganzzahlig/real (beide Variablentypen können angeschlossen werden)
- ..... Typ: Zeichenkette
- ..... Eingang - kein Kanal angeschlossen
- ..... Ausgang - kein Kanal angeschlossen
- ..... Eingang - mindestens ein Kanal angeschlossen
- ..... Ausgang - mindestens ein Kanal angeschlossen

Die folgenden Symbole werden benutzt, um den Typ der an einem Steckplatz untergebrachten E/A-Einrichtung anzuzeigen:

- ..... komplexe E/A-Baugruppe
- ..... reale E/A-Karte
- ..... virtuelle E/A-Karte

Die folgenden Symbole werden benutzt, um einen Parameter oder einen Kanal darzustellen:

- ..... Kartenparameter
- ..... freier Kanal
- ..... besetzter Kanal

Karten in der Liste verschieben

Benutzen Sie diese Schaltflächen in der Befehlsleiste oder die Befehle "**Bearbeiten / Karte nach oben/unten verschieben**", um die ausgewählte E/A-Karte in der Hauptliste zeilenweise nach oben oder nach unten zu verschieben. Der Befehl

"**Bearbeiten / Steckplatz einfügen**" fügt einen leeren Steckplatz an der aktuellen Position ein.

A.11.1. E/A-Karten definieren

Das Menü "**Bearbeiten**" enthält Befehle, um die ausgewählte Karte zu definieren (ihre Parameter zu bestimmen) und um die E/A-Variablen mit ihren Kanälen zu verdrahten.

Typ einer E/A-Karte auswählen

Bevor man E/A-Variablen an eine Karte anschließen kann, muß die Kartenkennzeichnung eingegeben werden. In der ISaGRAF Workstation ist eine Bibliothek mit vordefinierten Karten verfügbar. Diese Bibliothek wurde unter Umständen von einem oder mehreren E/A-Baugruppenherstellern kompiliert. Der Befehl "**Bearbeiten / Karte/Baugruppe bestimmen**" wird benutzt, um die Karte zu kennzeichnen. Mit diesem Befehl kann eine einzige Karte oder eine komplexe E/A-Baugruppe aus der ISaGRAF Bibliothek ausgewählt werden. Sie können auch auf einen Steckplatz doppelklicken, um die entsprechende Karte oder Baugruppe einzusetzen.
Sämtliche Kanäle einer einzelnen Karte haben den gleichen Typ (Boolesch, ganzzahlig/real oder Zeichenkette) und die gleiche Richtung (Eingang oder Ausgang). Zwischen realen und ganzzahligen Variablen wird bei der E/A-Verdrahtung nicht unterschieden. Eine komplexe E/A-Baugruppe ist eine E/A-Einrichtung mit Kanälen unterschiedlichen Typs oder unterschiedlicher Richtungen. Komplexe E/A-Baugruppen werden als eine Liste einzelner E/A-Karten dargestellt und belegen nur einen Steckplatz im Baugruppenträger.

Eine Karte entfernen

Der Befehl "**Bearbeiten / Steckplatz freigeben**" wird benutzt, um die ausgewählte Karte oder E/A-Baugruppe zu entfernen. Wenn bereits Variablen auf ihren Kanälen verdrahtet sind, werden sie automatisch freigegeben, wenn der Steckplatz freigegeben wird.

Reale und virtuelle Karten

Der Befehl "**Bearbeiten / Reale/virtuelle Karte**" bestimmt die Gültigkeit der ausgewählten Karte oder komplexen E/A-Baugruppe. Die folgenden Symbole erscheinen im Baugruppenträger, um die Gültigkeit einer Karte anzuzeigen:

 ..... reale E/A-Karte

 ..... virtuelle E/A-Karte

Im **Realmodus** sind die E/A-Variablen direkt mit den entsprechenden E/A-Einrichtungen verbunden. Die Ein- und Ausgangsoperationen im Anwendungsprogramm entsprechen unmittelbar den Ein- und Ausgangsbedingungen der wirklichen E/A-Einrichtungen. Im **Virtuellen Modus** werden die E/A-Variablen genauso wie interne Variablen verarbeitet. Sie können vom Debugger gelesen und aktualisiert werden, so daß der Benutzer die E/A-Verarbeitung simulieren kann, ohne daß ein wirklicher Anschluß erfolgt.

Technische Datenblätter

Mit dem Befehl "**Werkzeuge / Technisches Datenblatt**" kann die Herstelleranleitung der ausgewählten Karte oder Baugruppe online angezeigt werden. Das technische Datenblatt einer Karte wird vom Hersteller erstellt. Es enthält sämtliche Informationen bezüglich der Verwaltung einer Karte oder Baugruppe. Außerdem erläutert es die Bedeutung ihrer Parameter.

Angeschlossene Variablen entfernen

Der Befehl "**Werkzeuge / Kanäle freigeben**" unterbricht den Anschluß sämtlicher bereits mit der ausgewählten Karte verdrahteten E/A-Variablen.

Kommentare für freie Kanäle definieren

Der Kommentartext einer deklarierten E/A-Variablen wird zusammen mit ihrem Namen in der Kartenliste angezeigt. Da ISaGRAF auch die Benutzung von direkt dargestellten Variablen (% Notation) ermöglicht, können auch freien Kanälen Kommentare zugeordnet werden. Benutzen Sie den Befehl "**Werkzeuge / Kanalkommentar**", um einen Kommentar für einen freien, in der Kartenliste ausgewählten Kanal einzugeben. Dieser Befehl kann nicht benutzt werden, um den Kommentar einer im Datenverzeichnis des Projekts deklarierten E/A-Variablen zu modifizieren.

A.11.2. Kartenparameter bestimmen

Um den Wert eines Kartenparameters zu definieren, doppelklicken Sie auf seinen Namen in der rechten Liste. Sie können den Parameter auch auswählen (markieren) und den Befehl "**Kanal/Parameter definieren**" im Menü "**Bearbeiten**" einleiten. Die Parameter erscheinen am Anfang der Liste. Das folgende Symbol wird für die Darstellung eines Parameters in der Liste benutzt:

 ..... Kartenparameter

Die Bedeutung und das Eingangsformat eines Parameters werden vom Hersteller der entsprechenden E/A-Karte oder -Baugruppe geliefert. Benutzen Sie den Befehl "**Werkzeuge / Technisches Datenblatt**" oder lesen Sie das betreffende Hardware-Handbuch für weitere Informationen über die Parameter einer Karte.

A.11.3. E/A-Kanäle verdrahten

Um eine E/A-Variable mit einem Kanal zu verdrahten, doppelklicken Sie auf seine Position in der rechten Liste. Sie können den Kanal auch auswählen (markieren) und den Befehl "**Bearbeiten / Kanal/Parameter definieren**" einleiten. Die folgenden Symbole werden für die Darstellung der Kanäle in der Liste benutzt:

 ..... freier Kanal

 ..... besetzter (angeschlossener) Kanal

Die Liste enthält alle Variablen, die mit dem ausgewählten Kartentyp und der Richtung übereinstimmen. Die Liste enthält nur Variablen, die noch nicht verdrahtet sind. Die Schaltfläche "**Verdrahten**" verdrahtet die in der Liste ausgewählte Variable mit dem ausgewählten Kanal. Die Schaltfläche "**Freigeben**" entfernt die Variable vom ausgewählten Kanal. (Der Anschluß wird unterbrochen). Die Schaltflächen"**Nächster**" und "**Vorheriger**" werden benutzt, um einen anderen

Kartenkanal auszuwählen. Die Position des ausgewählten Kanals erscheint in der Titelzeile des Dialogfelds.

A.11.4. Direkt dargestellte Variablen

Unter freien Kanälen versteht man Kanäle, die nicht mit deklarierten E/A-Variablen verdrahtet sind. ISaGRAF ermöglicht die Benutzung von **direkt dargestellten Variablen** in den Quellcodes der Programme, um freie Kanäle darzustellen. Der Bezeichner einer direkt dargestellten Variablen beginnt stets mit dem Zeichen "**%**".
Es folgen die Benennungsregeln für eine direkt dargestellte Variable für den Kanal einer einzelnen Karte. "***s***" ist die Steckplatznummer der Karte. "***c***" ist die Nummer des Kanals.

%IXs.c ..... freier Kanal einer Booleschen Eingangskarte
%IDs.c ..... freier Kanal einer ganzzahligen Eingangskarte
%ISs.c ..... freier Kanal einer Zeichenketten-Eingangskarte
%QXs.c ..... freier Kanal einer Booleschen Ausgangskarte
%QDs.c ..... freier Kanal einer ganzzahligen Ausgangskarte
%QSs.c ..... freier Kanal einer Zeichenketten-Ausgangskarte

Es folgen die Benennungsregeln einer direkt dargestellte Variablen für den Kanal einer komplexen Baugruppe. "***s***" ist die Steckplatznummer der Baugruppe. "***b***" ist der Index einer einzelnen Karte innerhalb der Baugruppe. "***c***" ist die Nummer des Kanals.

%IXs.b.c ..... freier Kanal einer Booleschen Eingangskarte
%IDs.b.c ..... freier Kanal einer ganzzahligen Eingangskarte
%ISs.b.c ..... freier Kanal einer Zeichenketten-Eingangskarte
%QXs.b.c ..... freier Kanal einer Booleschen Ausgangskarte
%QDs.b.c ..... freier Kanal einer ganzzahligen Ausgangskarte
%QSs.b.c ..... freier Kanal einer Zeichenketten-Ausgangskarte

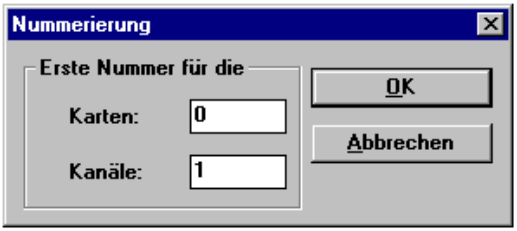
Hier einige Beispiele:

`%QX1.6` 6ter Kanal der Karte #1 (Boolescher Ausgang)
`%ID2.1.7` 7ter Kanal der Karte #1 in der Baugruppe #2 (ganzzahliger Eingang)

Eine direkt dargestellte Variable kann nicht den Datentyp "**real**" besitzen.

A.11.5. Nummerierung

Benutzen Sie den Befehl "**Optionen / Nummerierung**", um Nummerierungskonventionen zu definieren. Sie können die Nummer für den ersten und den letzten Steckplatz sowie die Nummer für den ersten Kanal einer jeden Karte im folgenden Dialogfeld eingeben:



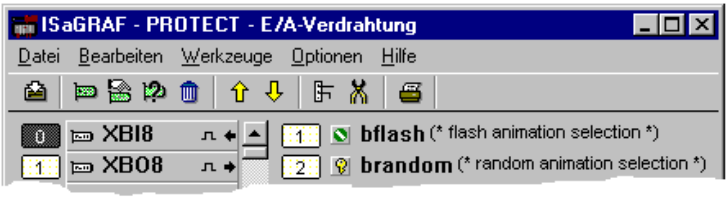
Als Vorgabe beginnt die Steckplatznummerierung bei Index **0** und die Kanalnummerierung bei Index **1**.

Achtung: Bei der Änderung der Nummerierungskonventionen ist äußerste Vorsicht geboten, denn sie nimmt Einfluß auf Symbole, die für direkt dargestellte Variablen benutzt werden. Es können Kompilierungsfehler entstehen, wenn direkt dargestellte Variablen in den bestehenden Programmen benutzt werden.

A.11.6. Individueller Datenschutz

Die ISaGRAF Workstation verfügt über ein komplettes Datenschutzsystem, das sich auf eine Paßwort-Hierarchie stützt. Die E/A-Verdrahtung kann global durch ein Paßwort geschützt werden. Zusätzlich ermöglicht ISaGRAF individuellen Schutz für einen beliebigen E/A-Kanal. Dies setzt voraus:
- daß schon Paßwörter im Paßwortsystem definiert wurden (Befehl "**Projekt** / **Paßwort**" im Fenster des Projektmanagers), so daß Datenschutzebenen für individuellen Datenschutz verfügbar sind.
- daß der individuelle Datenschutz auf einer höheren Prioritätsebene liegt als der globale Datenschutz.

Wenn ein E/A-Kanal individuell geschützt ist, erscheint ein kleines Symbol neben seinem Namen im Fenster der E/A-Verdrahtung:



Benutzen Sie die Befehle "**Datenschutz setzen**" und "**Datenschutz entfernen**" im Menü "**Bearbeiten**", um den individuellen Datenschutz des ausgewählten Kanals zu setzen oder zu entferenen. Beide Befehle erfordern die Eingabe eines gültigen Paßworts, damit die Zugriffsebene mit dem Kanal verknüpft werden kann. Daraufhin kann die Verdrahtung eines individuell geschützten Kanals nicht mehr ohne die Eingabe eines Paßworts einer genügend hohen Prioritätsebene modifiziert werden.

Achtung: Wenn ein Kanal durch eine Zugriffsebene geschützt ist und das entsprechende Paßwort aus dem Datenschutzsystem entfernt wird und wenn kein höheres Paßwort

definiert ist, kann die Verdrahtung des Kanals nicht mehr modifiziert werden, außer es wird ein neues Paßwort in einer genügend hohen Datenschutzebene definiert.

A.12. Umrechnungstabellen erstellen

Die ISaGRAF Workstation ermöglicht die Erstellung von Umrechnungstabellen. Eine Umrechnungstabelle besteht aus einer Gruppe von Punkten, die eine analoge Umrechnung definieren. Umrechnungstabellen können analogen Ein- und Ausgangsvariablen zugeordnet werden. Sie liefern die proportionelle Beziehung zwischen elektrischen (von Sensoren gelesenen oder zu Aktuatoren gesendeten) Werten und physikalischen (in Anwendungsprogrammen benutzten) Werten. Die Bearbeitung der Umrechnungstabellen erfolgt über ein Dialogfeld, das mit dem Befehl "**Werkzeuge / Umrechnung**" im ISaGRAF Datenverzeichnis-Fenster geöffnet wird.
Wenn eine Umrechnungstabelle definiert ist, kann sie benutzt werden, um die Werte einer beliebigen analogen Ein- oder Ausgangsvariablen des ausgewählten Projekts zu filtern. Um die Umrechnungstabelle einer Variablen zuzuordnen, benutzt man die Befehle des ISaGRAF Datenverzeichnisses (Editor für die Variablendeklarationen). Nun wählt man eine analoge Ein- oder Ausgangsvariable und bearbeitet deren Parameter. Umrechnungstabellen, die noch nicht definiert sind, können keinen Variablen zugeordnet werden.

A.12.1. Hauptbefehle

Die Liste der definierten Umrechnungstabellen erscheint im Dialogfeld "**Umrechnungstabellen**". Hier befinden sich auch die Schaltflächen für die hauptsächlichen Befehle: eine bestehenden Tabelle bearbeiten (Punkte definieren), eine neue Tabelle erstellen und eine Tabelle umbenennen oder löschen.
Klicken Sie OK, um das Dialogfeld "**Umrechnungstabellen**" zu schließen und die Tabellen auf der Festplatte zu speichern.

Eine neue Tabelle erstellen

Der Befehl "**Neu**" ermöglicht die Erstellung einer neuen Umrechnungstabelle. Es können bis zu **127** Umrechnungstabellen für jedes Projekt erstellt werden. Nur solche Tabellen, die benutzt werden (die analogen Variablen zugeordnet sind), werden auch in den Anwendungscode aufgenommen. Bei der Benennung der Umrechnungstabellen gelten die folgenden Regeln:

- Der Name muß sich auf **16** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Die weiteren Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Den Inhalt einer Tabelle ändern

Der Befehl "**Bearbeiten** " wird benutzt, um die Punkte einer in der Liste ausgewählten Umrechnungstabelle einzugeben. Man kann hierzu auch einfach auf einen Tabellennamen doppelklicken. Bei der Erstellung einer neuen Tabelle wird der Befehl "**Bearbeiten**" automatisch eingeleitet. Es müssen mindestens zwei Punkte pro Tabelle eingegeben werden.

A.12.2. Die Punkte einer Tabelle eingeben

Das Dialogfeld "**Bearbeiten**" dient der Eingabe der Tabellenpunkte. Das linke Feld zeigt die Liste der bereits definierten Punkte. Im Feld rechts unten wird die definierte Tabelle als Kurve abgebildet. Die Punkte werden mit Hilfe der Befehle des Dialogfelds eingegeben, wobei die Punktdefinitionsregeln beachtet werden müssen. Sie werden am Ende dieses Kapitels erläutert. Das linke Feld enthält stets die Liste der bestehenden Punkte für die in Arbeit befindliche Umrechnungstabelle. In der linken Spalte erscheinen die elektrischen (externen) Werte der Punkte. Die rechte Spalte zeigt die physikalischen (internen) Werte. Der Benutzer muß einen Punkt in der Liste auswählen, um dessen Werte zu ändern oder um den Punkt zu löschen (entfernen). Die letzte Auswahl in der Liste ("**... ...**") wird benutzt, um einen neuen Punkt zu definieren. Im Feld rechts unten wird die in Arbeit befindliche Tabelle als Kurve abgebildet. Achsen oder Koordinaten werden nicht angezeigt, da es sich um eine proportionelle Darstellung der Kurve handelt, die lediglich dazu dient, eventuelle Konstruktionsfehler aufzuzeigen.

Einen neuen Punkt definieren

Wenn ein neuer Punkt definiert werden soll, verlagert man die Auswahl auf das letzte Element in der Punktliste ("**... ...**"). Dieser Modus wird auch bei der Definition einer neuen Umrechnungstabelle vorgegeben. Nun werden die elektrischen (externen) und physikalischen (internen) Werte eines jeden Punktes eingeben. Die Werte werden als Gleitpunktzahlen mit einfacher Präzision gespeichert. Beachten Sie, daß mindestens **zwei Punkte** für die Definition einer Kurve erforderlich sind. Nach Eingabe der beiden Werte wird der Punkt durch Klicken auf die Schaltfläche "**Sichern**" in die Tabelle eingefügt. Es können höchstens **32** Punkte für eine Umrechnungstabelle definiert werden.

Einen Punkt modifizieren

Um die Werte eines bestehenden Punktes zu ändern, wird der Punkt in der Liste ausgewählt. Jetzt können die neuen elektrischen (externen) und physikalischen (internen) Werte für diesen Punkt eingegeben werden. Die Werte werden als Gleitpunktzahlen mit einfacher Präzision gespeichert. Nach Eingabe der beiden Werte klickt man auf die Schaltfläche "**Sichern**", um den Punkt in der Tabelle zu aktualisieren.

Einen Punkt löschen

Um einen bestehenden Punkt zu löschen, wird der Punkt in der Liste ausgewählt und die Schaltfläche "**Löschen**" gedrückt. Beachten Sie, daß mindestens **zwei Punkte** für die Definition einer Kurve erforderlich sind.

A.12.3. Regeln und Begrenzungen

Bei der Definition einer Umrechnungstabelle müssen die folgenden Regeln beachtet werden. Die Tabelle kann für die Umrechnung analoger Ein- und Ausgangsvariablen benutzt werden:

- Zwei Punkte dürfen nicht den gleichen elektrischen Wert haben.
- Die Kurve muß kontinuierlich steigend oder fallend sein.
- Zwei Punkte dürfen nicht den gleichen physikalischen Wert haben.

Die folgenden Begrenzungen gelten bei der Definition von Umrechnungstabellen für ein Projekt:

- Es können höchstens **127** Umrechnungstabellen für ein Projekt definiert werden.
- Es können höchstens **32** Punkte für eine Umrechnungstabelle definiert werden.

A.13. Codegenerator benutzen

Das Fenster des Codegenerators öffnet sich automatisch, wenn die Befehle "Prüfen" und "Codierung" der anderen ISaGRAF Fenster aktiviert werden. Das Fenster des Codegenerators wird nicht automatisch geschlossen, wenn die angeforderte Code-Entwicklung beendet ist, so daß der Benutzer über das Fenstermenü auch weiterhin auf alle Code-Entwicklungsbefehle und -optionen zugreifen kann.

A.13.1. Hauptbefehle

Das **"Datei"**-Menü enthält Befehle für die Prüfung der Programmsyntax und für die Code-Entwicklung.

Anwendungscode erstellen

Mit dem Befehl "**Codierung**" wird der gesamte Code des Projekts entwickelt. Vor der Code-Entwicklung prüft dieser Befehl die Syntax der Deklarationen und Programme. Fehler, die bei der Kompilierung einzelner Programme nicht erkannt wurden, werden bei der Code-Entwicklung sichtbar. Dies betrifft die Umrechnungstabellen, die Verdrahtung der E/A-Variablen und die Verknüpfungen mit den Bibliotheken. Die Code-Entwicklung stoppt die Kompilierung eines Programms, in dem Fehler erkannt werden. Das betreffende Programm muß korrigiert werden, bevor die Code-Entwicklung wieder aufgenommen werden kann. Bereits geprüfte (als fehlerfrei befundene) Programme, die seit ihrer letzten Prüfung nicht mehr geändert wurden, werden nicht erneut kompiliert. Die Variablendeklarationen und die Anwendungskonsistenz werden immer geprüft. Während der Programmprüfung kann die "**Codierung**" durch Drücken der **ESCAPE**-Taste abgebrochen werden.

Hinweis: Wenn die Deklaration einer programm-lokalen Variablen geändert wurde, wird das betreffende Programm geprüft. Wenn eine globale Variable geändert wurde, werden alle Programme geprüft.

Programmsyntax prüfen

Der Befehl "**Programm prüfen**" wird benutzt, um ein einzelnes Programm zu prüfen. Das ausgewählte Programm wird kompiliert, selbst wenn es seit seiner letzten Prüfung nicht geändert wurde. Der Befehl "**Datenverzeichnis prüfen**" wird benutzt, um die Deklarationen sämtlicher Variablen des Projekts zu prüfen.
Der Befehl "**Alle Programme prüfen**" prüft die Syntax aller Projektprogramme, selbst wenn manche Programme nicht geändert wurden. Diese Operation wird **nicht** unterbrochen, wenn ein Fehler in einem Programm erkannt wird. Der Befehl kann benutzt werden, um eine komplette Liste der noch in den Projektprogrammen vorhandenen Fehler aufzustellen. Der Befehl kann durch Drücken der **ESCAPE**-Taste abgebrochen werden.

Eine Änderung simulieren

Der Befehl "**Änderung simulieren**" simuliert die Änderung sämtlicher Projektprogramme, so daß sie alle bei der nächsten "**Codierung**" geprüft werden.

Der Befehl "**Öffnen**" wird benutzt, um das zuletzt geprüfte Programm zu öffnen. Dieser Befehl ist nützlich, um direkt auf ein Programm zuzugreifen, in dem Syntaxfehler erkannt wurden.

A.13.2. Compileroptionen

Mit dem Befehl "**Compiler-Optionen**" werden die hauptsächlichen Parameter bestimmt, die der ISaGRAF Codegenerator für die Erstellung und Optimierung des Zielcodes benutzt. Ziel dieses Befehls ist es, den zu generierenden Codetyp entsprechend der betreffenden Zielsysteme auszuwählen und die Optimierungsparameter entsprechend der erwarteten Kompilierungsdauer und den Anforderungen der Anwendungsausführung zu bestimmen.
Die Schaltfläche "**Rücklesen**" öffnet ein zweites Dialogfeld mit weiteren Optionen, die dem Einbetten von komprimiertem Quellcode im geladenen Code dienen, um "Rücklesen" zu ermöglichen. Siehe Dokumentation "Rücklesen" für Informationen zu dieser Funktionalität.

Zielsysteme auswählen

Die obere Liste zeigt die verfügbaren Zielcodes, die entwickelt werden können. Das Zeichen "**>>**" kennzeichnet, welche Zielcodes ausgewählt sind. Der ISaGRAF Codegenerator kann bis zu **3** verschiedene Codes in einer Kompilierungsoperation erstellen. Benutzen Sie die Schaltflächen "**Auswählen**" und "**Abwählen**", um die Liste der erforderlichen Zielcodes entsprechend Ihrer Ziel-Hardware zu bestimmen. Hier die ISaGRAF Standardzielcodes:

SIMULATE:..... Dieser Code ist dem ISaGRAF Simulator der Workstation dediziert. Der Simulator kann nicht starten, wenn dieser Zielcode nicht ausgewählt ist, um den Anwendungscode zu entwickeln.

ISA86M:..... Dies ist ein TIC-Code (Target Independent Code), der ISaGRAF Kernels dediziert ist, welche auf Intel-Prozessoren installiert sind. Der Prozessortyp betrifft nur die Byteordnung im entwickelten Code.

ISA68M:..... Dies ist ein TIC-Code (Target Independent Code), der ISaGRAF Kernels dediziert ist, welche auf Motorola-Prozessoren installiert sind. Der Prozessortyp betrifft nur die Byteordnung im entwickelten Code.

SCC:..... Die Auswahl dieses Zielcodes bewirkt, daß der ISaGRAF Compiler strukturierten "C"-Quellcode entwickelt, der kompiliert und mit den Bibliotheken des ISaGRAF Zielkernels verknüpft wird, um einen eingebetteten ausführbaren Code zu entwickeln

CC86M:..... Die Auswahl dieses Zielcodes bewirkt, daß der ISaGRAF Compiler nicht-strukturierten "C"-Quellcode entwickelt, der kompiliert und mit den Bibliotheken des ISaGRAF Zielkernels verknüpft wird, um einen eingebetteten ausführbaren Code zu entwickeln. Diese Wahl dient der Kompatibilität mit ISaGRAF Versionen unter V3.23, in der die strukturierte "C"-Code-Entwicklung nicht unterstützt wird.

Siehe Herstellerhandbuch Ihrer Hardware, um den Typ des auf Ihrer SPS installierten ISaGRAF Zielkernels ausfindig zu machen. Andere Zielcodes

(Maschinencode, C-Quellcode...) werden vorraussichtlich in zukünftigen Versionen der ISaGRAF Workstation unterstützt.

AS-Verarbeitung

Aktivieren Sie das Kontrollkästchen "**Eingebettetes AS-Engine benutzen**", um die Benutzung des ISaGRAF AS-Engine zu aktivieren. ***Dieser Modus sollte bevorzugt werden, da er verbesserte Betriebsleistungen erzielt***. Bei bestimmten Implementierungen des ISaGRAF Zielsystems kann das Ziel-Engine fehlen. Dies kommt jedoch häufiger bei kundenspezifischen Zielsystemen vor, die den ISaGRAF Code nachverarbeiten. In diesem Fall sollten Sie diese Option eventuell deaktivieren und es dem ISaGRAF Compiler überlassen, die AS-Netzwerke in einfache Anweisungen zu übersetzen. Siehe Herstellerhandbuch Ihrer Hardware für weitere Informationen zur Benutzung dieser Option.

Optimierungsoptionen

Die folgenden Parameter, die vom ISaGRAF Codegenerator benutzt werden, um den Zielcode zu optimieren, können im Dialogfeld "**Compiler-Optionen**" bestimmt werden. Die Schaltfläche "**Vorgabe**" wird benutzt, um alle Optimierungsoptionen zu entfernen und so die Kompilierungszeit zu verkürzen.

- Wenn die Option "**Zwei Durchläufe**" aktiviert wird, führt der ISaGRAF Code-Optimierer zwei Durchläufe aus. Die im zweiten Durchlauf ausgeführten Optimierungen sind gewöhnlich nicht so bedeutend wie die des ersten Durchlaufs.

- Wenn die Option "**Konstante Ausdrücke auswerten**" aktiviert wird, dann nimmt der Compiler eine Auswertung der konstanten Ausdrücke vor. Beispielsweise wird der numerische Ausdruck "**2 + 3**" im Zielcode durch "**5**" ersetzt. Wenn diese Option deaktiviert wird, dann werden die konstanten Ausdrücke bei der Ausführung berechnet.

- Wenn die Option "**Unbenutzte Marken unterdrücken**" aktiviert wird, vereinfacht der Optimierer das System der Sprünge und Marken des Programms, so daß unbenutzte Marken oder ziellose Sprünge unterdrückt werden.

- Wenn die Option "**Variablenkopien optimieren**" aktiviert wird, wird die Benutzung der temporären Variablen (die zur Speicherung von Zwischenergebnissen benutzt werden) optimiert. Diese Option wird gewöhnlich zusammen mit der Option "**Ausdrücke optimieren**" benutzt. Wenn diese Option aktiviert wird, werden die Ergebnisse von Ausdrücken und Unterausdrücken, die öfter als einmal im Programm benutzt werden, vom Optimierer wiederverwendet.

- Wenn die Option "**Unbenutzten Code unterdrücken**" aktiviert wird, wird unbedeutender Code vom Optimierer unterdrückt. Wenn beispielsweise die folgenden Anweisungen programmiert wurden: "**var := 1; var := X;**", ist der entsprechende Code lediglich: "**var := X;**".

- Wenn die Option "**Arithmetische Operationen optimieren**" aktiviert wird, vereinfacht der Optimierer die arithmetischen Operationen gemäß spezieller Operanden. Zum Beispiel: Der Ausdruck "**A + 0**" wird durch "**A**" ersetzt. Wenn die

Option "**Boolesche Operationen optimieren**" aktiviert wird, vereinfacht der Optimierer die Booleschen Operationen gemäß spezieller Operanden. Zum Beispiel: Der Boolesche Ausdruck "**A & A**" wird durch "**A**" ersetzt.

Wenn die Option "**Binäre Diagramme entwickeln**" aktiviert wird, ersetzt der Optmierer die Booleschen Gleichungen (mit gemischt vorkommenden **AND**, **OR** , **XOR** und **NOT** Operatoren) durch eine reduzierte Liste bedingter Sprungoperationen. Die Übersetzung erfolgt nur, wenn für die Sprungsequenz eine geringere Ausführungszeit als für den ursprünglichen Ausdruck erwartet wird.

Die folgende Tabelle ist eine Zusammenfassung der erwarteten Optimierung und Kompilierungsdauer für jeden Parameter:

	Leistungssteigerung	*Kompilierungsdauer*
2 Durchläufe	**xxxx**	(*)
Konstante Ausdrücke auswerten	**xxxxxxxxxx**	**xxxx**
Unbenutzte Marken unterdrücken	**xxxx**	**xxxxxxxxxx**
Variablenkopien optimieren	**xxxx**	**xxxxxxxxxx**
Ausdrücke optimieren	**xxxx**	**xxxxxxxxxx**
Unbenutzten Code unterdrücken	**xxxx**	**xxxxxxxxxx**
Arithmetische Operationen opt.	**xxxxxxxxxx**	**xxxx**
Boolesche Ausdrücke optimieren	**xxxxxxxxxx**	**xxxx**
Binäre Diagramme entwickeln	**xxxxxxxxxxxxxxx**	**xxxxxxxxxxxxxxx**

(*) die Kompilierungsdauer wird verdoppelt.

A.13.3. C-Quellcode

Mit der ISaGRAF Workstation kann Quellcode in der Hochsprache "C" entwickelt werden. In diesem Fall wird der gesamte Inhalt einer Anwendung, einschließlich AS-Netzwerkbeschreibung, Datenbankdefinition und Codesequenzen, im "C"-Quellcodeformat generiert. Es gibt zwei Möglichkeiten:

CC86M (C-Quellcode - V3.04) entwickelt nicht-strukturierten "C"-Quellcode. Dieser Stil sollte gewählt werden, wenn Ihre Zielsystem-Software auf ISaGRAF Versionen unter 3.23 basiert.

SCC (strukturierter C-Quellcode) entwickelt strukturierten C-Quellcode. Dieser Stil sollte gewählt werden, wenn Ihre Zielsystem-Software auf ISaGRAF Version 3.23 oder höher basiert.

Die folgenden zwei Dateien werden im Projektverzeichnis erstellt:
APPLI.C ..... allgemeiner Quellcode der Anwendung
APPLI.H ..... allgemeine Definitionen der "C"-Sprache

Wenn strukturierter "C"-Quellcode entwickelt wird, wird eine "**.C**"-Quelldatei und eine "**.H**"-Definitionsdatei für jedes Programm der Anwendung (zusätzlich zu den allgemeinen Dateien "**APPLI.C**" und "**APPLI.H**") erstellt.

Diese Dateien müssen kompiliert und mit den ISaGRAF Zielbibliotheken verknüpft werden, um den endgültigen ausführbaren Code zu entwickeln. Siehe

Benutzerhandbuch des "ISaGRAF I/O Development Toolkit " für ausführliche Informationen zu den empfohlenen Implementierungsmethoden.

Hinweis: Manche Debugging-Funktionalitäten, wie Laden der Anwendung, Online-Änderungen und Haltepunkte sind nicht mehr verfügbar, wenn die ISaGRAF Anwendung in "C" kompiliert wird.

A.13.4. Informationen visualisieren

Mit den Befehlen im Menü "**Bearbeiten**" können die verschiedenen Textdateien, die bei der Code-Entwicklung und Syntaxprüfung erstellt wurden, im Fenster des Codegenerators visualisiert werden. Dieses Fenster ist ein Textfeld, welches während der Code-Entwicklung und Syntaxprüfung Meldungen anzeigt. Sämtliche Informationen werden auf der Festplatte gespeichert, so daß sie mit den Menübefehlen "**Bearbeiten**" eingesehen werden können.

Bearbeitungsbefehle

Der Befehl "**Bildschirmtext löschen**" wird benutzt, um den Text im Fenster zu löschen. Der Fenstertext wird automatisch vor jeder Code-Entwicklung und Syntaxprüfung gelöscht. Der Befehl "**Kopieren**" wird benutzt, um den angezeigten Text in der Windows-Zwischenablage zu kopieren, so daß er von anderen Anwendungen, z.B. den ISaGRAF Texteditoren, benutzt werden kann.

Kompilierungsmeldungen visualisieren

Der Befehl "**Ausführungsmeldungen**" visualisiert sämtliche Meldungen, die während der letzten Codierungs- und Prüfoperationen im Textfeld angezeigt wurden. Dies betrifft alle Fehlermeldungen.

Andere Befehle im Menü "**Bearbeiten**" ermöglichen, Hilfstextdateien einzusehen, die während der Code-Entwicklung und Syntaxprüfung erstellt wurden. Diese Dateien werden normalerweise nicht für ein gewöhnliches ISaGRAF Projekt benutzt.

A.13.5. Resourcen definieren

Der Befehl "**Resourcen**" im Menü "**Optionen**" ermöglicht die Definition von Resourcen. Resourcen sind anwender-definierte Daten (Netzwerk-Konfiguration, Hardware-Einstellung...) beliebigen Formats (Datei, Liste von Werten), die mit dem generierten Code verbunden werden müssen, damit sie mit ihm auf die Ziel-SPS geladen werden können. Solche Daten werden nicht direkt vom ISaGRAF Kernel verarbeitet und sind gewöhnlich einer anderen, auf der Zielsteuerung installierten, Software dediziert. Siehe Herstellerhandbuch Ihrer Hardware für weitere Informationen über verfügbare Resourcen.

Die Resourcen-Definitionsdatei

Die Resourcen werden in einer "**Resourcen-Definitionsdatei**" gespeichert, die mit den anderen Dateien des ISaGRAF Projekts gespeichert wird. Es handelt sich um eine reine ASCII-Textdatei, die vom ISaGRAF Resourcen-Compiler verarbeitet wird. Dieser Compiler wird automatisch gestartet, wenn der Anwendungscode entwickelt wird. In diesem Abschnitt wird die Syntax dieser Datei erläutert. Die Resourcen-

Definitionsdatei benutzt die lexikalischen Regeln der ST-Sprache. Kommentare, die mit dem Zeichen"**(***" anfangen und mit mit dem Zeichen "***)**" enden, können an beliebigen Stellen im Text eingefügt werden. Zeichenketten werden von einfachen Apostrophen begrenzt. Siehe Teil 2 dieses Handbuchs für weitere Informationen über die lexikalischen Formate, die für die Eingabe von numerischen Werten benutzt werden.

Sprachreferenzen

Es folgt eine Liste der Schlüsselwörter und Anweisungen, die in der Resourcen-Definitionsdatei benutzt werden.

ULONGDATA

Bedeutung: Spezifiziert eine Resource, die eine Liste ganzzahliger Werte ist. Die Werte werden im Zielcode als vorzeichenlose Ganzzahlen auf 32 Bits gespeichert. Die Werte werden in einer in der Resourcen-Definitionsdatei spezifizierten Ordnung gespeichert. Die Werte müssen durch Kommas getrennt werden. Der Name der Resource darf 15 Zeichen nicht überschreiten.

Syntax:
```
ULONGDATA '<resource_name>'
BEGIN
  ...target_selection...
  ...list of values...
END
```

Beispiel:
```
ULongData 'MYDATA'
Begin
  ...
  0, -1, 100_000,        (* dezimal *)
  16#A0B1, 2#1011_0101   (* hexadezimal, binär *)
End
```

VARLIST

Bedeutung: Spezifiziert eine Resource, die eine Liste von Variablenadressen ist. Die Variablen werden durch ihren Namen in der Resourcen-Definitionsdatei identifiziert. Die Variablenadressen werden im Zielcode als vorzeichenlose Ganzzahlen auf 16 Bits gespeichert. Die Adressen werden in einer in der Resourcen-Definitionsdatei spezifizierten Ordnung gespeichert. Die Variablen müssen durch Kommas getrennt werden. Der Name der Resource darf 15 Zeichen nicht überschreiten.

Syntax:
```
VARLIST '<resource_name>'
BEGIN
  ...target_selection...
  ...list of variable names...
END
```

Beispiel:
```
VarList 'LIST'
```

```
Begin
  ...
  Var100, MyParameter, Command, Alarm
End
```

BINARYFILE

Bedeutung: Spezifiziert eine Resource vom Typ Binary File. Die Ursprungsdaten werden in einer MS-DOS Datei gespeichert. Die Definition der Zielresource wird mit einem Zielpfadnamen vervollständigt. Zeilenendzeichen werden nicht vom ISaGRAF Resourcen-Compiler konvertiert. Der Name der Resource darf **15** Zeichen nicht überschreiten.

Syntax:
```
BINARYFILE '<resource_name>'
BEGIN
  ...target selection...
  FROM '<urspungs_pfadname>'
  TO '<ziel_pfadname>'
END
```

Beispiel:
```
BinaryFile 'MYFILE'
Begin
  ...
  From 'c:\user\config.bin'
  To '/dd/user/appl/config.dat'
End
```

TEXTFILE

Bedeutung: Spezifiziert eine Textdatei-Resource. Die Ursprungsdaten werden in einer ASCII-Datei gespeichert. Die Definition der Zielresource wird mit einem Zielpfadnamen vervollständigt. Die Zeilenendzeichen werden vom ISaGRAF Resourcen-Compiler gemäß der Konventionen des Zielsystems konvertiert. Der Name der Resource darf **15** Zeichen nicht überschreiten.

Syntax:
```
TEXTFILE '<resource_name>'
BEGIN
  ...target selection...
  FROM '<urspungs_pfadname>'
  TO '<ziel_pfadname>'
END
```

Beispiel:
```
TextFile 'MYFILE'
Begin
  ...
  From 'c:\user\config.bin'
  To '/dd/user/appl/config.dat'
End
```

TARGET

Bedeutung: Spezifiziert den Namen eines Zielcodes, der die Resource einschließen muß. Siehe vorangehenden Abschnitt (Compiler-Optionen) für weitere Informationen über die verarbeiteten Zielcodes. Die Anweisung "**Target**" kann mehrmals in einem Resourcen-Block auftreten, damit mehrere Zielcodes ausgewählt werden können. Diese Anweisung kann nicht benutzt werden, wenn die Anweisung "**AnyTarget**" spezifiziert ist.

```
Syntax:    TARGET '<zielcode_name>'

Beispiel:  BinaryFile 'MYFILE'
           Begin
            Target 'ISA86M'
            Target 'ISA68M'
            ...
           End
```

ANYTARGET

Bedeutung: Spezifiziert, daß die Resource mit allen vom Code-Generator entwickelten Zielcodes verbunden werden muß. Der ISaGRAF Code-Generator kann mehrere Zielcodes in einem **Codierung**svorgang entwickeln. Die Anweisung kann nicht benutzt werden, wenn eine oder mehrere "**Target**"-Anweisungen spezifiziert sind.

```
Syntax:    ANYTARGET
Beispiel:  ULongData 'MYDATA'
           Begin
            AnyTarget
            ...
           End
```

FROM

Bedeutung: Spezifiziert den Quellpfadnamen (auf dem PC, auf dem die ISaGRAF Workstation installiert ist) einer Resource vom Typ **BinaryFile** oder **TextFile**. Die Trennzeichen für die Komponenten des Pfadnamens (Laufwerk, Verzeichnis, Vorzeichen, Suffix) müssen konform zu den MS-DOS-Systemkonventionen sein.

```
Syntax:    FROM '<target pathname>'

Beispiel:  BinaryFile 'MYFILE'
           Begin
            ...
            From 'c:\user\config.dat'
            To '/dd/user/appl/config.dat'
           End
```

TO

Bedeutung: Spezifiziert den Zielpfadnamen (auf dem Zielsystem) einer Resource vom Typ **BinaryFile** oder **TextFile**. Die Trennzeichen für die Komponenten des Pfadnamens (Laufwerk, Verzeichnis, Vorzeichen, Suffix) müssen konform zu den Ziesystemkonventionen sein.

Syntax: **TO '**<target pathname>**'**

```
Beispiel: TextFile 'MYFILE'
          Begin
            ...
            From 'c:\user\config.dat'
            To '/dd/user/appl/config.dat'
          End
```

Beispiel

Es folgt das komplette Beispiel einer Resourcen-Definitionsdatei:

```
(* Resourcen-Definitionsdatei *)

ULongData 'DATA1'             (* Liste von Werten *)
Begin
   Target 'ISA86M'            (* nur für diesen Zielcode *)
   1, 0, 16#1A2B3C4D, +1, -1 (* numerische Werte *)
End

VarList 'VLIST1'              (* Variablenliste *)
Begin
   Target 'ISA86M'            (* nur für diesen Zielcode *)
   Valve1, StateX, Command, Alrm1 (* Variablennamen *)
End

BinaryFile 'FILE1'            (* binary file Resource *)
Begin
   AnyTarget                  (* allen Zielcodes dediziert *)
   From 'c:\user\updatef.bin' (* Quelldatei auf PC *)
   To 'updatef.cfg'           (* Zieldatei auf SPS *)
End

TextFile 'FILE2'              (* Textdatei-Resource *)
Begin
   Target 'ISA68M'
   From 'c:\nw\nwbd.txt'      (* Quelldatei auf PC *)
   To '/nw/dat/nwbd'          (* Zieldatei auf SPS *)
End
```

Resourcen kompilieren

Wenn Resourcen in die Resourcen-Definitionsdatei eingegeben wurden, erscheint das "Resourcen kompilieren" Dialogfeld am Ende der ISaGRAF Code-Entwicklung.

Klicken Sie auf die Schaltfläche "**Kompilierung starten**", um den Resourcen-Compiler zu starten. Ausführungs- und Fehlermeldungen werden im der Kontrolltafel angezeigt. Klicken Sie "**Beenden**", wenn Sie die Resourcen nicht kompilieren wollen. In diesem Fall werden die Resourcen dem ISaGRAF Code nicht beigefügt.

Implementierung

Unter ISaGRAF ist die Anzahl der Resourcen, sowie die Größe der Datenzeilen und Dateien unbegrenzt. Resourcen werden am Ende des generierten Codes, mit einem Resourcen-Verzeichnis, gespeichert. Es folgt das Format des Resourcen-Verzeichnisses (mit C-Notationen):

```
__RESOURCE:
{
  long nbres;             /* Anzahl definierter Resourcen */
  {
    char name[16];        /* Resourcen-Name */
    long type;            /* Resourcen-Datentyp */
    long size;            /* genaue Größe des Datenblocks */
    uint32 data;
    uint32 path_offset;   /* Pointer zu einer Zeichenkette */
  } /*Anzahl der Datensätze */
}
```

Das Feld "type" kann die folgenden Werte annehmen:
- 1 = binary file
- 2 = text file
- 3 = ulong data (in diesem Fall wird das Feld path_offset nicht benutzt)
- 4 = variable list (in diesem Fall wird das Feld path_offset nicht benutzt)

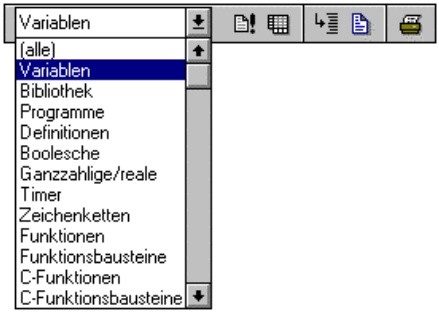
Für Textdateien werden die Zeilenendzeichen vom Resourcen-Compiler gemüß der Zielsystemkonventionen übersetzt. Alle Pointers sind 32 Bit Offsets von der Adresse der entsprechenden Struktur. Alle Resourcen- und Pfadnamen sind in NULL endende Zeichenketten. Pfadnamen und Datenfluß folgen dem Resourcen-Verzeichnis.

A.14. Crossreferencen

Die ISaGRAF Workstation verfügt über einen Crossreferenzen-Editor, der eine Übersicht über das Vorkommen der deklarierten Variablen in den Programmen des Projekts vermittelt. Ziel der Crossreferenzen ist es, alle im Projekt deklarierten Variablen aufzulisten und sie in den Quellcodes der Programme zu lokalisieren. Die Crossreferenzen sind nützlich, wenn eine globale Übersicht über den Lebenszyklus einer Variablen benötigt wird. Sie ermöglichen das Aufspüren von Nebenwirkungen und machen Anwendungen bei der Instandhaltung verständlicher. Die Crossreferenzen können auch für eine globale Übersicht über das gesamte Datenverzeichnis eines Projekts benutzt werden, so daß unbenutzte Variablen leichter gefunden werden und die Komplexität des Projekts beurteilt werden kann.
Die Liste links im Fenster zeigt die deklarierten Objekte (Programme, Variablen und Definitionen) des Projekts, sowie die im Projekt referenzierten Bibliothekselemente (Funktionen und Funktionsbausteine). Die Liste rechts im Fenster zeigt die Vorkommen in den Programmen des momentan in der linken Liste ausgewählten Objekts.
Die Beschreibung eines Vorkommens schließt den Programmnamen, die Nummer des Schritts, der Transition oder des Tests (FD oder AS), sowie die Zeilennummer (bei Textsprachen) oder die Koordinaten (bei KOP- oder FBS-Diagrammen) ein. Bei schnellen KOP-Diagrammen wird außerdem die Rung-Nummer angezeigt. Wenn die Variable als Ausgang (auf einer Spule) benutzt wird, folgt ein Stern ("*****") auf die Rung-Nummer.
Aktivieren Sie die Option "**Unbenutzte Variablen anzeigen**" im Menü "**Optionen**", um in der Hauptliste auch die Variablen anzuzeigen, die nicht in den Anwendungsprogrammen benutzt werden.

Auswahl des Objekttyps

Projekte können eine große Anzahl an deklarierten Objekten enthalten. Die Anzeige kann deshalb auf bestimmte Objekttypen begrenzt werden. Die anzuzeigenden Objekttypen werden in einem Listenfeld in der Editor-Befehlsleiste ausgewählt:



Bei jeder neuen Berechnung der Crossreferenzen wird die Auswahl auf "**Alle**" zurückgesetzt und die vollständige Liste angezeigt.

Crossreferenzen neu berechnen

Der Befehl "**Datei / Neu berechnen**" kann jederzeit benutzt werden, um die Crossreferenzen mit den Modifikationen zu aktualisieren, die in anderen ISaGRAF Fenstern eingegeben wurden.

Crossreferenzen exportieren

Der Befehl "**Werkzeuge / Exportieren**" wird benutzt, um die komplette Liste der Crossreferenzen in eine ASCII-Textdatei zu exportieren. Diese Datei kann dann mit anderen Anwendungen, wie Windows NotePad oder Textverarbeitungsprogrammen, geöffnet werden.

Fehler im Datenverzeichnis

Der Befehl "**Bearbeiten / Datenverzeichnisfehler**" zeigt eine Fehlerliste in einem Dialogfeld an. Hier erscheinen die Fehler, die beim Laden des Datenverzeichnisses eines Projekts erkannt wurden.

Statistiken

Der Befehl "**Werkzeuge / Statistik**" zeigt die Zahl der in einem Projekt deklarierten Objekte und Variablen in einem Dialogfeld an (nach Variablentypen und -attributen). Dieser Befehl ist besonders nützlich, wenn man mit einer begrenzten Version der ISaGRAF Workstation arbeitet und feststellen möchte, wieviele E/A-Variablen in einem Projekt deklariert wurden, d.h. ob das Projekt kompiliert werden kann.

In der Objektliste suchen

Der Befehl "**Bearbeiten / Suchen**" ermöglicht die direkte Auswahl eines Objekts in der Liste. Wenn das gesuchte Objekt allerdings nicht dem angezeigten Objekttyp entspricht, dann kann es auch nicht gefunden werden. Deshalb wird empfohlen, vor der Objektsuche die Option "**Alle Objekte**" in der Befehlsleiste zu aktivieren.

Programm öffnen

Die Liste rechts im Fenster enthält die Vorkommen eines ausgewählten Objekts in den Quellcodes und der E/A-Verdrahtung des offenen Projekts. Mit dem Befehl "**Bearbeiten / Programm öffnen**" kann ein Programm geöffnet werden, in dem das Objekt erscheint. Man kann auch mit der Maus auf das Vorkommen eines Objekt doppelklicken (in der Liste), um das entsprechende Programm zu öffnen.

A.15. Grafischen Debugger benutzen

Die ISaGRAF Workstation verfügt über einen vollständigen, grafischen und symbolischen Debugger. Der Befehl "Debuggen" im Fenster des Programm-Managers startet den Debugger, um die auf die Ziel-SPS geladene Anwendung zu testen. In diesem Modus erfolgt ein Dialog mit der Zielsteuerung über einen seriellen Anschluß. Der Befehl "Simulieren" im Fenster des Programm-Managers startet den Debugger und die Simulation der Zielsteuerung gleichzeitig. Auf diese Weise können Anwendungen getestet werden, wenn die Zielsteuerung noch nicht verfügbar ist. Die Befehle im Debugger-Fenster dienen der Kontrolle der gesamten Anwendung.
Wenn der Debugger startet und wenn die Anwendung in der Zielsteuerung die gleiche wie die in der Workstation ist, wird das Fenster des **Programm-Managers** automatisch im Debugging-Modus geöffnet. Die Befehle dieses Fensters können jetzt benutzt werden, um die anderen ISaGRAF Fenster (Grafik- und Texteditoren, Datenverzeichnis, Variablenlisten, E/A-Verdrahtung...) zu öffnen. Alle während einer Debugging-Sitzung geöffneten Fenster befinden sich im "**Debugging-Modus**", d.h. die Bearbeitungsbefehle sind deaktiviert. Die angezeigten Programmkomponenten (Schritte, Transitionen, Variablen...) erscheinen mit ihren aktuellen Ausführungszuständen oder -werten. Wenn Sie auf eines dieser Objekte doppelklicken, können Sie seinen Zustand oder Wert in der Zielanwendung ändern.
Wenn der Debugger im **Simulations-Modus** benutzt wird, erfolgt keine Kommunikation mit dem ISaGRAF Zielsystem. Der Debugger dialogiert lediglich mit dem Simulator-Fenster. Da das Zielsystem in diesem Modus nicht existiert, sind die Befehle "**Übertragen**", "**Steuerung stoppen**" und "**Steuerung starten**" im Debugger-Menü nicht verfügbar.

A.15.1. Debugger-Fenster

Im Fenster des Debuggers werden nur solche Daten angezeigt, die über den Status der gesamten Anwendung Auskunft geben. Dieses Fenster ist mit den anderen ISaGRAF Fenstern verknüpft, wodurch ein vollständiges, interaktives Debugging-System geschaffen wird. Erkannte Ausführungsfehler werden im unteren Teil des Debugger-Fensters angezeigt. Mit den Befehlen im Menü "**Optionen**" kann die Fehlerliste angezeigt, nicht angezeigt oder gelöscht werden.
Die Kontrolltafel (unterhalb des Debugging-Menüs) zeigt den globalen Status der Zielanwendung an und gibt Auskunft über die Zykluszeit. Folgende Meldungen werden für die Anzeige des Status der Zielanwendung benutzt:

Anschluß:.....Der Debugger etabliert die Kommunikation mit dem Zielsystem.

Unterbrochen:.....Der Debugger kann nicht mit dem Zielsystem dialogieren. Prüfen Sie das Anschlußkabel und die Kommunikationsparameter.

Keine Anwendung: .....Der Anschluß ist OK, aber es befindet sich keine ISaGRAF Anwendung im Zielsystem. Laden Sie eine Anwendung.

Anwendung aktiv:.....Der Anschluß ist OK und es befindet sich eine aktive Anwendung im Zielsystem. Der Debugger etabliert jetzt

den Dialog mit dieser Anwendung, wenn es sich um die gleiche Anwendung wie die in der Workstation handelt.

RUN: .....Die Zielanwendung befindet sich im "Echtzeitmodus".
STOP: .....Die Zielanwendung befindet sich im "Schrittmodus".
Haltepunkt:.....Die Anwendung befindet sich im "Schrittmodus", denn es wurde ein Haltepunkt angetroffen.
Fataler Fehler:.....Die Zielanwendung ist wegen eines fatalen Fehlers funktionsuntüchtig.

Folgende Anzeigen geben Auskunft über die Zykluszeit:

Zugelassen: .....Programmierte Zykluszeit.
Aktuell: .....Dauer des letzten vollständigen Ausführungszyklus.
Maximum:.....Längste Zykluszeit seit Anwendungsstart.
Überschreitung:.....Anzahl erkannter Ausführungszyklen, die mehr Zeit als zugelassen in Anspruch genommen haben.

Alle Zeitwerte werden in Millisekunden angezeigt. Diese Zeitwerte werden nicht angezeigt, wenn der Debugger im Simulationsmodus benutzt wird.

A.15.2. Anwendungen steuern

Die Menüs "**Datei**" und "**Kontrolle**" enthalten Befehle für die Installation und Kontrolle der aktuellen ISaGRAF Anwendung auf dem ISaGRAF Zielsystem.
Anmerkung: Manche dieser Befehle sind im Simulationsmodus nicht verfügbar, da die vom Simulator verarbeitete Anwendung automatisch von der ISaGRAF Workstation installiert wird.

Zielanwendung stoppen

Der Befehl "**Datei / Steuerung stoppen**" stoppt die Ausführung der im ISaGRAF Zielsystem aktiven Anwendung.

Zielanwendung aktivieren

Der Befehl "**Datei / Steuerung starten**" aktiviert die Anwendung im Zielsystem. Nach dem Laden einer Anwendung startet diese automatisch und der Befehl "**Starten**" braucht nicht benutzt zu werden. Der Befehl "**Starten**" wird insbesondere nach einem "**Stop**"-Befehl benutzt.
Hinweis: Die Zielanwendung muß gestoppt (inaktiv) sein, bevor eine neue Anwendung geladen werden kann.

Anwendung laden

Der Befehl "**Datei / Übertragen**" wird benutzt, um den Anwendungscode in das Zielsystem zu laden. Wählen Sie den zu ladenden Code entsprechend des Zielsystem-Prozessors und der Anwendungsoptionen.

Versionsnummer anzeigen

Der Befehl "**Datei / Versionsnummer lesen**" wird benutzt, um die komplette Identifizierung der Workstation- und Zielanwendungen anzuzeigen. Die Workstation-Anwendung ist die, welche gerade auf der ISaGRAF Workstation offen ist. Die

Zielanwendung ist die, welche in der ISaGRAF Ziel-SPS ausgeführt wird. Folgende Elemente werden angezeigt:

VERSION: .....Versionsnummer des Anwendungscodes (vom Codegenerator berechnet)
DATUM: .....Datum und Uhrzeit der Code-Entwicklung
CRC: .....Checksum, das vom Codegenerator anhand des Inhalts der Symboltabelle berechnet wurde. Dieser Wert hängt vom Inhalt des Variablen-Datenverzeichnisses ab.

Hinweis: Der Befehl "**Versionsnummer lesen**" ist auch im Simulationsmodus verfügbar. Im Debugging-Modus kann dieser Befehl nur benutzt werden, wenn die Ziel-SPS angeschlossen ist.

Online-Änderungen

Mit dem Befehl "**Datei / Anwendung aktualisieren**" kann der Benutzer "Online-Änderungen" auf der laufenden Zielanwendung ausführen. Dieser Befehl wird weiter unten in diesem Kapitel erläutert. Er ist nicht verfügbar, wenn der Debugger im Simulationsmodus benutzt wird.

Echtzeitmodus

Der Befehl "**Kontrolle / Echtzeitmodus**" ist nur verfügbar, wenn eine Anwendung aktiv ist. Er setzt die Zielanwendung auf den normalen "Echtzeitmodus". Normaler Modus: Die Ausführungszyklen werden von der programmierten Zykluszeit ausgelöst.

Schrittmodus

Der Befehl "**Kontrolle / Schrittmodus**" ist nur verfügbar, wenn eine Anwendung aktiv ist. Er setzt die Zielanwendung auf den "Schrittmodus": In diesem Modus werden die Zyklen einzeln ausgeführt, wenn der Benutzer den Befehl "**Einen Zyklus ausführen**" im Debugging-Menü einleitet.

Einen Zyklus ausführen

Wenn sich die Zielanwendung im Schrittmodus befindet, bewirkt der Befehl "**Kontrolle / Einen Zyklus ausführen**" die Ausführung eines Zyklus.

Zykluszeit

Mit dem Befehl "**Kontrolle / Zykluszeit ändern**" kann der Benutzer die programmierte Zykluszeit ändern. Diese Zeit erscheint unter der Anzeige "**Zugelassen**" in der Kontrolltafel des Debuggers. Vor der Änderung der Zykluszeit sollte der "**Schrittmodus**" eingeleitet werden. Die Zykluszeit wird als ganzzahliger Wert in Millisekunden eingegeben.

Alle Haltepunkte entfernen

Der Befehl "**Kontrolle / Alle Haltepunkte löschen**" löscht alle aktuellen (angetroffenen und noch aktiven) Haltepunkte der gesamten Anwendung. Bestehende Haltepunkte werden nicht automatisch gelöscht, wenn das Debugger-Fenster geschlossen wird.

E/A-Variablen freigeben

Der Befehl "**Kontrolle / Alle E/A-Variablen freigeben**" gibt sämtliche verriegelten E/A-Variablen der Anwendung wieder frei. Wenn eine E/A-Variable verriegelt ist, werden keine Zustandsänderungen an den Ein- und Ausgängen der entsprechenden E/A-Einrichtungen vorgenommen. Variablen, die mit den E/A verknüpft sind, können noch von der Anwendung oder dem Debugger forciert werden. Verriegelte E/A-Variablen werden beim Schließen des Debugger-Fensters nicht automatisch freigegeben.

A.15.3. Optionen

Das Menü "**Optionen**" enthält Optionen für die Anzeigen im Debugger-Fenster:

- ***Kommunikationsparameter***

  Wenn der Debugger aktiv ist, können nur zeitbedingte Kommunikationsparameter definiert werden (**Time-Outs**). Die anderen Kommunikationsparameter (Geschwindigkeit, Parität...) werden vom Menü "**Debuggen**" im Fenster des Programm-Managers aus definiert.
  Das "**Time-Out der Kommunikation**" ist die Zeit, die dem Zielsystem gewährt wird, bevor es beginnt, eine Anfrage der Workstation zu beantworten. Die "**Aktualisierungsperiode**" ist die vom Debugger benötigte Zeit, um "**Leseanfragen**" zu senden, die der Aktualisierung der offenen Fenster dienen.
  Alle Zeitwerte werden als ganzzahlige Werte in **Millisekunden** eingegeben und angezeigt. Die zeitbedingten Kommunikationsparameter können nicht bestimmt werden, wenn der Debugger im Simulationsmodus benutzt wird.

- ***Anzeigeoptionen***

  Die Option "**Zykluszeit anzeigen**" ermöglicht die Anzeige oder Nicht-Anzeige der **Zykluszeitwerte** in der Debugger-Kontrolltafel. Wenn diese Option aktiviert wird, werden sämtliche Zykluszeitkomponenten (Zugelassen, Aktuell, Maximum, Überschreitungen) angezeigt und aktualisiert. Durch Deaktivieren dieser Option wird der Arbeitsaufwand des Debuggers für die Kommunikation reduziert.
  Wenn die Option "**Fehler anzeigen**" aktiviert wird, werden die bei der Ausführung erkannten Fehler laufend im unteren Teil des Debugger-Fensters angezeigt. Wenn diese Option deaktiviert wird, dann bleibt die Fehlerliste geschlossen und die Kommunikations- und Anzeigearbeit des Debuggers wird reduziert. Der Befehl "**Optionen / Fehler löschen**" löscht den Inhalt der aktuellen Fehlerliste im Debugger-Fenster.
  Der Befehl "**Optionen / Fenstergröße reduzieren**" verkleinert das Debugger-Fenster, so daß es als eine kleine Tafel angezeigt wird, in der nur der Anwendungsstatus und die grafischen Schaltflächen für die geläufigsten Befehle angezeigt werden.

A.15.4. "Schreib"-Befehle

Der symbolische ISaGRAF Debugger enthält zahlreiche Befehle, mit denen die **Werte** oder **Zustände** der Anwendungskomponenten modifiziert werden können. Um ein Element auszuwählen, **doppelklicken** Sie auf seinen Namen oder sein Symbol in einem der Bearbeitungsfenster, wenn das Debugger-Fenster offen ist.

Variablen

Der Zustand einer Variablen kann geändert werden, indem man auf ihren Namen in einem der folgenden Fenster doppelklickt:

- Datenverzeichnis
- Variablenlisten oder Steuerungsdiagramme
- KOP- oder FBS-Programme
- E/A-Verdrahtung

Die folgenden Befehle sind im Debugging-Dialogfeld verfügbar:

- Variable zu einem neuen Wert forcieren
- Variable **verriegeln** (nur für E/A-Variablen)
- Variable **freigeben** (nur für verriegelte E/A-Variablen)
- Timer-Variable **starten** oder **stoppen** (automatischen Aktualisierungsmodus setzen)

Die symbolischen Werte, die benutzt werden, um die Booleschen Werte **FALSE** und **TRUE** darzustellen, sind Zeichenfolgen, die für diese spezifische Boolesche Variable im Datenverzeichnis definiert wurden. Ein analoger Wert, der für einen **Forcierung**sbefehl spezifiziert wird, muß in einem ganzzahligen oder realen Format erfaßt werden, entsprechend der Variablendefinition im Datenverzeichnis. Eine Zeichenfolge, die für einen **Forcierung**sbefehl einer Zeichenketten-Variablen spezifiziert wird, darf die bei der Deklaration dieser Variablen bestimmte, maximale Länge nicht überschreiten.

AS-Objekte

Um ein **AS-Programm** in der Debugging-Phase zu steuern, werden die Befehle des "**Datei**"-Menüs im Fenster des Programm-Managers benutzt. Das AS-Programm wird in der Programmliste ausgewählt. Die folgenden Befehle sind verfügbar:

AS-Programm starten: ..... Aktiviert das ausgewählte Programm (setzt eine AS-Zugriffsberechtigung in jeden seiner Initialisierungsschritte)

AS-Programm stoppen: ..... Stoppt das ausgewählte Programm (entfernt alle bestehenden Zugriffsberechtigungen).

AS-Programm anhalten:..... Entfernt alle bestehenden Zugriffsberechtigungen des ausgewählten Programms und speichert deren Positionen.

AS-Programm erneut starten:... Startet ein angehaltenes Programm erneut, indem alle durch den Befehl "anhalten" entfernten Zugriffsberechtigungen wiederhergestellt werden.

Bei Sohnprogrammen entsprechen diese Befehle den Funktionen "**GSTART**", "**GKILL**", "**GFREEZE**" und "**GRST**" der Programmiersprache.
Um einen **AS-Schritt** in der Debugging-Phase einer Anwendung zu steuern, doppelklicken Sie auf sein grafisches Symbol im AS-Bearbeitungsfenster. Die folgenden Befehle sind im Debugging-Dialogfeld verfügbar:

- Haltepunkt auf die **Aktivierung** eines Schritts setzen
- Haltepunkt auf die **Deaktivierung** eines Schritts setzen
- Gesetzten Haltepunkt eines Schritts **löschen**

Hinweis: Aktivierungs- und Deaktivierungs-Haltepunkte können nicht auf den gleichen Schritt gesetzt werden.

Um eine **AS-Transition** in der Debugging-Phase einer Anwendung zu steuern, doppelklicken Sie auf ihr grafisches Symbol im AS-Bearbeitungsfenster. Die folgenden Befehle sind im Debugging-Dialogfeld verfügbar:

- **Haltepunkt** auf die Überschreitung einer Transition setzen
- Gesetzten Haltepunkt der Transition **löschen**
- Transition manuell **überschreiten** (Zugriffsberechtigungen verschieben oder hinzufügen)

Bedingtes Überschreiten: Eine Zugriffsberechtigung wird in allen einer Transition folgenden Schritten erstellt. Die Zugriffsberechtigungen in den vorangehenden Schritten werden gelöscht.
Unbedingtes Überschreiten: Eine Zugriffsberechtigung wird in allen einer Transition folgenden Schritten erstellt. Die Zugriffsberechtigungen in den vorangehenden Schritten werden nicht gelöscht.

A.15.5. Online-Änderungen

Mit Hilfe der Funktion "Online-Änderungen" kann eine aktive Anwendung während der Prozeßausführung vom Benutzer modifiziert werden. Bei manchen chemischen Prozessen, bei denen eine Prozeßunterbrechung die Produktion oder sogar die Sicherheit beeinträchtigen könnten, ist diese Vorgehensweise unumgänglich. Online-Änderungen sollten **mit äußerster Vorsicht** vorgenommen werden, da ISaGRAF unter bestimmten Umständen nicht alle Konflikte erkennen kann, die durch solche benutzerdefinierten Online-Änderungen entstehen könnten.

Codesequenzen

Da ISaGRAF über zahlreiche Möglichkeiten verfügt, vom Debugger aus auf Variablen, Programme oder E/A-Karten zuzugreifen, bezieht sich die hier beschriebene Funktion "Online-Änderungen" nur auf die Codesequenzen. Eine Codesequenz ist ein kompletter Satz an ST-, AWL-, KOP- oder FBS-Anweisungen, welche in einer Reihenfolge ausgeführt werden. In Programmen der Abschnitte "Zyklusanfang" und "Zyklusende" ist eine Codesequenz die gesamte Liste der Anweisungen eines Programms. In AS-Programmen ist eine Codesequenz die Programmierung der Ebene 2 eines Schrittes oder einer Transition. In einer "Online-Änderung" werden eine oder mehrere Codesequenzen ersetzt, ohne daß der SPS-Ausführungszyklus gestoppt wird. Da die Steuerung der AS-Zugriffsberechtigungen äußerst kritisch ist, **ist es nicht möglich, die AS-Struktur zu ändern oder Schritte, Transitionen oder AS-Programme hinzuzufügen, neu zu nummerieren oder zu löschen.**

Variablen

Die Variablen-Datenbank ist ebenfalls ein äußerst kritischer Teil einer Anwendung und andere Prozesse können jederzeit auf sie zugreifen (bei Multitasking-SPS). Zudem können die Variablenwerte vom Debugger aus modifiziert werden. Deshalb ist es unter **ISaGRAF nicht möglich, Variablen online hinzuzufügen, umzubenennen oder zu löschen**. Allerdings kann die Art, wie eine Variable in der Anwendung benutzt wird, geändert werden. Es ist auch möglich, "unbenutzte"

interne oder E/A-Variablen in der ersten Version einer Anwendung zurückzulegen, so daß sie in zukünftigen Änderungen benutzt werden können.

In der ISaGRAF Zielsystem-Datenbank gibt es Variablen verschiedenen Stils. Bei allen gibt es bestimmte Einschränkungen:

- Deklarierte Variablen

Dies sind Variablen, die mit dem ISaGRAF Datenverzeichnis deklariert wurden. Sie können nicht geändert oder für Online-Änderungen umbenannt werden. Es wird empfohlen, zusätzliche Variablen zu deklarieren und in der Anwendung zu initialisieren, selbst wenn sie nicht sofort gebraucht werden. Solche zusätzlichen Variablen ermöglichen zukünftige Änderungen, ohne das Daten-Checksum der Anwendung zu modifizieren.

- Funktionsbaustein-Instancen

Jede Instance eines in "C" oder IEC geschriebenen Funktionsbausteins entspricht Daten, die in der Echtzeit-Datenbank des ISaGRAF Zielsystems gespeichert werden. Wenn Funktionsbaustein-Instancen hinzugefügt oder entfernt werden, sind keine Online-Änderungen mehr möglich. Deshalb is es besser, in ST (mit den im Datenverzeichnis deklarierten Funktionsbaustein-Instancen) zu arbeiten, als Bausteine in Schnellen KOP- oder FBS-Diagrammen hinzuzufügen, welche neuen, automatische deklarierten Instancen entsprechen. Auch die Modifizierung der Definition eines in der Bibliothek verfügbaren Funktionsbausteins macht eine Online-Änderung unmöglich.

- Schritte

Jeder AS-Schritt entspricht Daten, in denen die dynamischen AS-Schrittattribute (Aktivitätszeit und Flagge) gespeichert sind. Das Hinzufügen oder Entfernen von AS-Schritten ändert die Datenbank der Anwendung und ist bei Online-Änderungen nicht erlaubt.

- Versteckte, vom Compiler zugeordnete Variablen

Der ISaGRAF Compiler generiert "versteckte" Zeitvariablen, um komplexe Ausdrücke zu lösen. In manchen Fällen kann die Änderung eines solchen Ausdrucks zu einem unterschiedlichen Satz unsichtbarer Zeitvariablen führen und eine Online-Änderung unmöglich machen. Um diese Situation zu vermeiden, können der ISA.INI Datei die folgenden Eingaben beigefügt werden, um jedem Programm eine festgelegte Anzahl von Zeitvariablen zuzuordnen, selbst wenn diese nicht für die Kompilierung der ersten Version der Anwendung benötigt werden. Die folgenden Werte sind Beispiele:

```
[DEBUG]
MNTVboo=8     ; für Boolesche Variablen
MNTVana=4     ; für ganzzahlige und reale Variablen
MNTVtmr=4     ; für Timer-Variablen
MNTVmsg=2     ; für Zeichenketten-Variablen
```

Wenn diese Eingaben in ISA.INI erscheinen, gibt der Compiler eine Warnmeldung aus, falls die Neukompilierung der Anwendung eine höhere Anzahl an zugeordneten Zeitvariablen nach sich zieht.

Eingänge und Ausgänge

Da das ISaGRAF E/A-System ausgesprochen offen gestaltet ist, sollten die notwendigen Änderungen von einem OEM, unter Berücksichtigung der speziellen Funktionalitäten der betreffenden Hardware, implementiert werden. Das ISaGRAF System **ermöglicht nicht, E/A-Variablen online hinzuzufügen, anzuschließen oder freizugeben oder die Beschreibung einer E/A-Karte online zu ändern**. Die Modifizierung der Kartenparameter und die Verriegelung der E/A-Kanäle erfolgt mit Hilfe von Standard-OEM-Funktionalitäten und der Funktion "**OPERATE**".

Ausführungsoperationen

Um eine in Betrieb befindliche Anwendung zu modifizieren, werden die folgenden Operationen ausgeführt:

- Quellcode der Anwendung in der Workstation modifizieren
- Neuen Anwendungscode generieren
- Neuen Anwendungscode mit Hilfe des Befehls "**aktualisieren**" laden (anstatt des Befehls "**übertragen**")
- Zwischen SPS-Ausführungszyklen, mit Hilfe des Befehls "**Aktualisierung ausführen**", von der alten Anwendung zur neuen Anwendung übergehen.

Diese Vorgehensweise garantiert, daß die Ziel-SPS stets über eine komplette, zuverlässige und in Betrieb befindliche Anwendung verfügt und ermöglicht dem Benutzer, die zeitlichen Vorgänge der jeweiligen Operationen auf sichere und wirksame Weise zu steuern. Außerdem kann ein Projekt so oft wie möglich geändert werden. Mit Ausnahme eventueller Konsequenzen für den Prozeß selbst ist die "Online-Änderung" einem normalen Satz an "**Stop-**, **Start-** und **Ladebefehlen**" sehr ähnlich. Der einzige Unterschied ist, daß die Zustände der Variablen nicht verlorengehen und daß die Übergangszeit von einer Anwendung zur anderen sehr kurz ist (im allgemeinen 1 oder 2 Zyklen). Während des Übergangs werden keine Variablen modifiziert und **alle internen, Eingangs- und Ausgangsvariablen** behalten die gleichen Werte vor und nach der Änderung der Anwendung bei. Während des Übergangs werden keine Aktionen ausgeführt und die **AS-Zugriffsberechtigungen werden nicht bewegt**.

Speicheranforderungen

Um "Online-Änderungen" unterstützen zu können, benötigt die Ziel-SPS ausreichend freien Speicherplatz, damit die modifizierte Version des Anwendungscodes gespeichert werden kann, denn während des Übergangs vom alten zum neuen Anwendungscode werden beide Codes im Speicher der SPS gespeichert.

Begrenzungen

Wie bereits erwähnt sind ausschließlich Änderungen der Codesequenzen gestattet. Variablendefinitionen, Anwendungsparameter und die E/A-Verdrahtung können nicht modifiziert werden. Wenn die neue Version einer Anwendung geladen wird, wird sie von ISaGRAF mit der in Betrieb befindlichen Anwendung verglichen, um eventuelle gefährliche Änderungen zu erkennen. Wenn die Übertragung als zu

riskant oder als unmöglich befunden wird, wird ein Übertragungsfehler generiert. Eine der von ISaGRAF ausgeführten Kontrollen ist der Vergleich der Checksums der Symboltabellen, so daß Änderungen der Variablennamen, Programme oder AS-Elemente erkannt werden können. Wenn ein Schritt während des Übergangs aktiv ist, so gehen seine nicht-gespeicherten Aktionen (N) verloren. Die neuen Schrittaktivierungsaktionen werden nicht ausgeführt. Aktionen, die bei der Deaktivierung des Schritts ausgeführt werden, werden vom neuen Anwendungscode übernommen. Wenn eine Transition gültig ist, wenn der Übergang stattfindet, wird ihre Rezeptivitätsgleichung aktualisiert. Der neu geladene Anwendungscode wird in der SPS nicht gesichert. Die Sicherungskopie enthält die Version, welche vorher mit den Standardladebefehlen geladen wurde.

Operationen

Um den Code einer in Betrieb befindlichen Anwendung zu aktualisieren, werden die folgenden Operationen ausgeführt:

- Bevor man Änderungen an einer laufenden Anwendung vornimmt, wird empfohlen, das aktuelle Projekt unter einem anderen Namen zu kopieren. Die Änderungen können an den Kopien vorgenommen werden.
- Bevor ein Programm bearbeitet wird, sollte man prüfen, ob die Option "**Protokolldatei aktualisieren**" in den Bearbeitungswerkzeugen aktiviert ist, um zukünftige Instandhaltungen des Programms zu erleichtern.
- Wenn eine oder mehrere Sequenzen modifiziert wurden (ohne die AS-Struktur oder die Programmhierarchie zu verändern), muß der Code vor dem Laden in der Workstation generiert werden.
- Unter Anwendung des Debuggers muß der Benutzer die Ziel-SPS vom alten Projekt aus anschließen und alle Operationen ausführen, welche zu einer schnelleren und zuverlässigeren Aktualisierung der Anwendung beitragen.
- Unter Anwendung des Debuggers muß der Benutzer die Ziel-SPS vom neuen Projekt aus anschließen. Wenn der Name der Anwendung geändert wurde, ist kein Zugriff auf die Zieldatenbank möglich. Der Benutzer muß den Befehl "**Datei / Aktualisieren**" einleiten.
- Die modifizierte Anwendung wird durch Auswahl der Option "**Später aktualisieren**" übertragen (geladen). Hierdurch wird die SPS bei der Übertragung eventuell etwas verlangsamt.
- Nach beendeter Übertragung kann der Benutzer den Befehl "**Datei / Aktualisierung ausführen**" einleiten, um den Übergang im günstigsten Moment zu ermöglichen. Der Übergang nimmt 1 bis 2 Zyklen in Anspruch.
- Wenn die Übertragung ordnungsgemäß ausgeführt wurde, werden die Programme der neuen, nun in Betrieb befindlichen Anwendung angezeigt. Wenn nicht, bleibt die ursprüngliche Anwendung aktiv.

A.15.6. DDE-Datenaustausch

Der ISaGRAF Debugger verfügt über einen DDE-Server (Dynamic Data Exchange). Zwischen dem ISaGRAF Debugger und anderen Anwendungen kann eine Mitteilungsringleitung (advise loop) installiert werden, so daß die aktuellen Variablenwerte in nicht-ISaGRAF Anwendungen dynamisch angezeigt werden.
Ausschließlich Transaktionen vom Typ "advise" und "poke" werden vom DDE-Server des ISaGRAF Debuggers unterstützt. Transaktionen vom Typ "request" können nur für Variablen benutzt werden, die bereits in einer Mitteilungsringleitung

überwacht werden. Andere DDE-Leistungen, wie "execute", sind nicht verfügbar. Wenn eine Mitteilungsringleitung auf einer Variablen erstellt ist, wird der Wert der Variablen jedesmal in der Client-Anwendung aktualisiert, wenn die Variable sich verändert. Beliebige Variablentypen können überwacht werden. Die Identifizierung der dynamischen Verknüpfung schließt die folgenden Namen ein:

Service-Name:	"ISaGRAF"
Name des Dokuments:	Name des ISaGRAF Projekts
Name des Elements:	Name der Variablen

Wenn es sich um eine programm-lokale Variable handelt, muß ihrem Namen der eingeklammerte Name ihres Vaterprogramms folgen, und zwar in der folgenden Syntax:

variablen_name(programm_name)

Der DDE-Server des ISaGRAF Debuggers ist der aktuellen im Debugger überwachten ISaGRAF Anwendung dediziert. Bis zu **256** Variablen können vom ISaGRAF Server überwacht werden. Der DDE-Server kann sowohl im Onlinemodus des Debuggers als auch im Simulationsmodus benutzt werden. Die Aktualisierungsperiode ist die gleiche, wie die, welche für die Kommunikation zwischen dem Debugger und dem ISaGRAF Zielsystem oder Simulator definiert wurde.

A.16. Variablenlisten überwachen

Der Befehl "**Listen beobachten**" im Menü "**Spion**" des Debugger-Fensters wird benutzt, um Variablenlisten zu erstellen, die mit ihren aktuellen Werten aktualisiert werden. Diese Listen werden beim Debuggen einer Anwendung erstellt, können auf der Festplatte gespeichert und bei anderen Debugging-Sitzungen wieder geöffnet werden. Eine Liste kann bis zu **32** Variablen unterschiedlichen Typs enthalten. Globale und lokale Variablen können gemischt in einer Liste vorkommen. Variablenlisten sind einem bestimmten Projekt dediziert. Sie sind besonders für die Funktionstests einer Anwendung von Nutzen und ermöglichen die Visualisierung und Überwachung eines beschränkten Teils des Steuerungsprozesses, unabhängig vom entsprechenden Quellcode in den Anwendungsprogrammen. Variablenlisten sind aber auch beim Debugging von ST- und AWL-Textprogrammen von Nutzen. Der Benutzer kann einen Satz an Programmvariablen problemlos in einer Liste zusammenfassen, um die Ausführung der programmierten Anweisungen zu beobachten und zu steuern.
Für jede Variable der Liste zeigt ISaGRAF den Namen, den aktuellen Wert und den Kommentartext an. Die Spalten können vergrößert oder verkleinert werden, indem man die betreffenden Trennlinien in der Listentitelzeile mit der Maus verlagert.

Listen auf der Festplatte sichern

Die Befehle im "**Datei**"-Menü werden benutzt, um Variablenlisten zu erstellen, zu öffnen und zu sichern. Unter ISaGRAF ist die Anzahl der Variablenlisten für ein Projekt unbegrenzt. Bei der Benennung der auf der Festplatte zu sichernden Variablenlisten müssen die folgenden Regeln beachtet werden:

- Der Name muß sich auf **8** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Weitere Zeichen können **Buchstaben**, **Zahlen** oder '_' sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Der Listeneditor kann nicht mehrere Variablenlisten in einem Fenster anzeigen. Es ist jedoch möglich, mehrere Fenster mit unterschiedlichen Variablenlisten zu öffnen, um verschiedene Listen gleichzeitig auf dem Bildschirm zu beobachten.

Variablen in die Liste einfügen

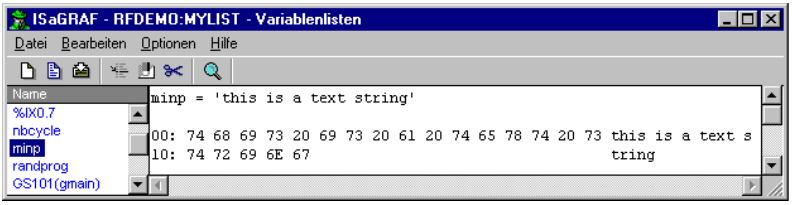
Der Befehl "**Bearbeiten / Einfügen**" fügt eine neue Variable in die Liste ein. Der Variablenname kann in der Liste der im Datenverzeichnis des Projekts deklarierten Objekte ausgewählt werden, so daß der Bezeichner nicht manuell eingegeben werden muß. Die neue Variable wird vor der in der Liste markierten Variablen eingefügt. Eine Liste kann nicht mehr als **32** Variablen enthalten. Eine Variable kann nicht mehrmals in einer Liste erscheinen.

Die ausgewählte Variable ändern

Der Befehl "**Bearbeiten / Ändern**" ersetzt die ausgewählte Variable durch eine andere Variable. Sie können auch den Befehl "**Ausschneiden**" benutzen, um eine ausgewählte Variable aus der Liste zu nehmen.

Auszug anzeigen

Die Listenvisualisierung kann jederzeit auf die Anzeige eines "Auszugs" umgeschaltet werden. Klicken Sie auf die "Zoom"-Schaltfläche in der Befehlsleiste oder benutzen Sie den Befehl "**Optionen / Auszug**" um zwischen Visualisierungsmoden zu wechseln.
Im "Auszugsmodus" wird nur ein Variablenwert angezeigt. Der Wert erscheint oben im Fenster im numerischen/symbolischen Format und wird außerdem im binären."Auszugsformat" angezeigt. Dieser Modus ermöglicht die Anzeige des Hexadezimalwerts eines jeden Bytes des Variablenwerts.



Die Anzeige eines "Auszugs" ist sehr nützlich, wenn Zeichenketten, die nicht-druckbare Zeichen enthalten, überwacht und verstanden werden sollen.

A.17. ST- und AWL-Programme debuggen

Während des Simulations- oder Online-Debugging eines ST- oder AWL-Programms können keine Änderungen am Programmtext vorgenommen werden.

AWL Bei AWL-Programmen erscheinen die Anweisungen in einer Listenübersicht. Der aktuelle Wert einer in einer Anweisung benutzten Variablen wird auf derselben Zeile angezeigt. Durch Doppelklicken auf eine Anweisung kann der Wert der zugeordneten Variablen geändert werden.

ST Bei ST-Programmen ist ein Überwachungslisten-Fenster in das Editorfenster eingebettet. Die Größe der Anzeigefelder kann verändert werden, indem man die Trennline durch Ziehen mit der Maus verlagert.

Jede Variable in der Liste wird mit ihrem Namen, laufenden Wert und Kommentartext angezeigt. Die Spaltenbreite kann verändert werden, indem man die Trennlinien in der Titelzeile der Liste durch Ziehen mit der Maus verlagert.

Listen auf der Festplatte sichern

Mit dem Befehl "**Datei / Liste sichern**" wird die Variablenliste auf der Festplatte unter dem gleichen Namen wie das bearbeitete Programm gesichert. Diese Liste wird automatisch jedesmal neu geladen, wenn das ST- oder AWL-Programm im Debugging-Modus geöffnet wird. Die Liste kann auch mit dem Überwachungslisten-Werkzeug, das mit dem Befehl "**Überwachen / Überwachungsliste**" im Debugger-Fenster eingeleitet wird, geöffnet und modifiziert werden.

Variablen in die Liste einfügen

Mit dem Befehl "**Bearbeiten / Variable einfügen**" wird ein Variable in die Liste eingefügt. Der Variablenname wird aus der im Datenverzeichnis des Projekts definierten Objektliste ausgewählt und braucht somit nicht manuell vom Benutzer eingegeben zu werden. Die Variable wird vor der in der Liste ausgewählten Variablen eingefügt. Die Liste kann maximal **32** Variablen enthalten. Eine Variable kann nicht mehrmals in einer Liste erscheinen.

Wenn sich ein Variablenname im ST-Text abhebt, klicken Sie auf diese Schaltfläche in der Toolbox oder leiten Sie den Befehl "Bearbeiten / Überwachen" ein, um die Variable direkt in die eingebettete Überwachungsliste zu senden.

Die ausgewählte Variable ändern

Der Befehl "**Bearbeiten / Variable ändern**" ersetzt die ausgewählte Variable durch eine andere Variable. Sie können auch den Befehl "**Variable ausschneiden**" benutzen, um die ausgewählte Variable aus der Liste zu nehmen.

A.18. Debuggen mit SpotLight

Mit dem ISaGRAF Werkzeug "SpotLight" kann der Benutzer Überwachungslisten erstellen, die beim Debugging als Grafiken oder Listen angezeigt werden können. Die Variablen eines ISaGRAF Projekts werden grafischen Elementen zugeordnet. Die Grafiken werden "online" definiert und bewegt.
Um den Wert einer Variablen zu forcieren, doppelklickt man auf das betreffende Element im grafischen Layout oder im Listenlayout oder drückt EINGABE, wenn das Element ausgewählt ist.
Das Dokument kann mit Hilfe des Befehls "**Datei / Verriegeln**" verriegelt (gegen Änderungen geschützt) werden. Auch in einem verriegelten Dokument können die Variablen weiterhin durch Doppelklicken auf ihre Symbole forciert werden.

A.18.1. Das grafische Layout

Es wird ein Diagramm aus Hintergrundbildern (Bitmaps oder Metafiles) und einem Satz grafischer Elemente erstellt, die sich beim Debugging bewegen. Für die Erstellung des Diagramms sind die folgenden Operationen erforderlich: Einfügen der Hintergrundbilder, Einfügen der grafischen Elemente und die Verbindung der Objekte mit den Projektvariablen.

Hintergrundbilder

Die Hintergrundbilder sind Dateien vom Typ "Bitmap" (.BMP) oder “Metafile” (.WMF). Die Anzahl der Bilder im grafischen Layout ist unbegrenzt. (Im Listenlayout erscheinen sie nicht). Die Bilder können verschoben oder vergrößert und verkleinert werden. Sie werden mit anderen Werkzeugen erstellt, da SpotLight über keine eigenen Zeichenwerkzeuge verfügt. Der Befehl "**Optionen / Hintergrundfarbe**" wird benutzt, um eine Hintergrundfarbe für den freien Platz im grafischen Layout zu definieren.

Hinweis: Bitmaps beanspruchen sehr viel Speicherplatz. Es wird strengstens empfohlen, die Größe eines Bildes dahingehend zu verändern, daß sich der unbenutzte Platz im Inneren des Bitmap-Rechtecks auf ein Minimum begrenzt.

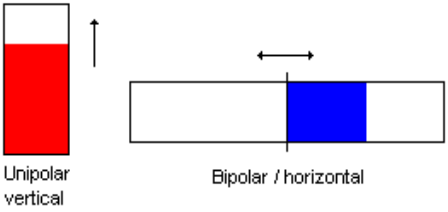
Einfache Textanzeige

Die “einfache Textanzeige” besteht aus einem Text, der in ein Rechteck geschrieben wird. Der angezeigte Text ist der Wert der zugeordneten Variablen. Daher ist es möglich, ein solches Element mit einer Zeichenketten-Variablen zu verbinden.
Das Textrechteck kann farbig oder transparent gestaltet werden. Wenn die Größe des Elements geändert wird, paßt die Schrift sich dem Rechteck an.

Unipolare und bipolare Histogramme

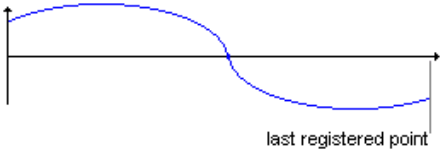
Ein Histogramm ist ein Rechteck mit einem farbigen Teil, der den numerischen Wert der zugeordneten Variablen darstellt. Der verbleibende Teil des Rechtecks kann wahlweise mit Farbe aufgefüllt werden. Histogramme können horizontal oder vertikal verlaufen.
Unipolare Histogramme können sich nach allen Seiten ausdehnen: nach oben, nach unten, nach links oder nach rechts. Bipolare Histogramme können sich in positiven

oder negativen Richtungen ausdehnen, dem Wert der zugeordneten Variablen entsprechend. Bei einem bipolaren Histogramm gilt der definierte Höchstwert gleichermaßen für die negativen und die positiven Werte.



Kurven

Es besteht die Möglichkeit, eine Kurve in ein Dokument einzufügen, welche die Entwicklung der zugeordneten Variablen anzeigt. Obwohl eine derartige Kurve keinen genauen Messungen entspricht, kann sie beim Debugging nützliche Informationen über die Synchronisierung verschiedener Variablen liefern.
Eine Kurve speichert die letzten 200 Werte einer Variablen. Die Anzahl der Samples bleibt gleich, selbst wenn man das Kurvenelement im grafischen Layout vergrößert oder verkleinert.

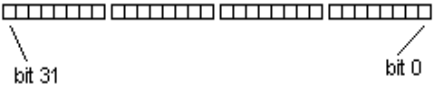


Boolesche Symbole

Das “Boolesche Symbol" wird benutzt, um einen binären Zustand anzuzeigen. Man definiert eine Symboldatei (.ICO) für den Wert FALSE oder 0 und eine weitere Symboldatei für alle anderen Werte ungleich null. Da SpotLight über keinen Symboleditor verfügt, müssen die Symboldateien mit einem anderen Werkzeug erstellt werden.

Bitfelder

Ein “Bitfeld” ist ein grafisches Feld, in dem die 32 Bits eines ganzzahligen Werts angezeigt werden. Das niederwertigste Bit erscheint immer rechts im Feld. Bitfelder sollten nur für ganzzahlige Werte benutzt werden. Bei anderen Werten, zum Beipiel realen analogen Werten, könnten die angezeigten Daten zu Verwechselungen führen.



Elemente auswählen, verschieben, ihre Größe verändern

Die Auswahl grafischer Objekte ist für die meisten Bearbeitungsbefehle erforderlich. SpotLight ermöglicht die Auswahl eines oder mehrerer Objekte im Diagramm. Für die Objektauswahl muß die Schaltfläche "**Auswahl**" (Schaltfläche mit Abbildung eines Pfeils) in der Toolbox des Editors "gedrückt" werden. Um ein Objekt auszuwählen, klickt man einfach auf sein Symbol. Um eine Liste von Objekten auszuwählen, zieht man die Maus im Zeichenfeld, um ein Rechteck zu markieren. Alle grafischen Objekte, die von diesem Auswahlrechteck durchkreuzt werden, werden als "**ausgewählt**" markiert, d. h. sie erscheinen mit kleinen schwarzen Kästchen auf ihrem Rahmen.
Bei einer neuen Auswahl wird die vorherige Auswahl nichtig. Um eine bestehende Auswahl aufzuheben, klickt man einfach mit der Maus in den leeren Raum außerhalb des Auswahlrechtecks.
Wenn ein Objekt verschoben werden soll, muß es zuerst ausgewählt werden. Dann verlagert man den Cursor auf den Rahmen des ausgewählten Elements und zieht es an eine andere Stelle.
Wenn ein Objekt vergrößert oder verkleinert werden soll, muß es zuerst ausgewählt werden. Man plaziert den Mauscursor auf eins der kleinen schwarzen Kästchen auf seinem Rahmen und zieht den Rahmen in die gewünschte Richtung, um die Größe des Objekts zu verändern. Man kann auch die Größe eines Bildes verändern. In diesem Fall verändert das entsprechende Bitmap oder Metafile seine Form, um sich dem neuen Rechteck anzupassen.

Elemente gruppieren / Gruppen trennen

Einzelne Elemente können gruppiert werden, so daß sie wie ein einziges Element verwaltet werden können. Um Elemente zu gruppieren, wählt man die Elemente im grafischen Layout aus und leitet den Befehl "**Bearbeiten / Gruppieren**" ein. Der Befehl "**Bearbeiten / Trennen**" wird benutzt, um die Elemente wieder zu trennen, so daß sie erneut einzeln bearbeitet werden können.
Eine Gruppe kann ein Bild beinhalten. Ferner kann sie andere Gruppen einschließen.
Wenn Elemente gruppiert sind, kann ihr Stil nicht mehr geändert werden. Die Elemente der Gruppe werden noch angezeigt, können aber nicht mehr angeklickt werden, um die Werte der zugeordneten Variablen zu modifizieren.
Im Listenlayout erscheint eine Gruppe auf einer einzigen Zeile.

A.18.2. Das Listenlayout

Es ist jederzeit möglich, mit Hilfe dieser Schaltfläche vom grafischen Layout zum Listenlayout überzuwechseln. Man kann auch den Befehl "**Optionen / Liste - Grafik**" benutzen.
Im Listenlayout werden die Elemente in einem klassischen Listenfeld angezeigt. Die Höhe eines jeden Elements wird entsprechend seines Zeichenstils berechnet. Bilder (Bitmaps und Metafiles) sind im Listenlayout nicht sichtbar. Der im Listenlayout verfügbare Auswahlmodus wird benutzt, um den Stil eines Elements oder den Wert

einer Variablen zu verändern. Die Auswahl mehrerer Elemente und die dazugehörigen Befehle sind im Listenlayout nicht verfügbar.

Die Elemente der Liste können mit dem Befehl "**Bearbeiten / In Liste verschieben**" neu geordnet werden. Das zu verschiebende Element muß in der Liste ausgewählt werden.

A.18.3. Elementstil definieren

Um den grafischen Stil und die Einstellungen eines bestehenden Elements zu modifizieren, doppelklickt man auf sein Symbol in der Grafik oder leitet den Befehl "**Bearbeiten / Stil**" ein, wenn das Element im grafischen Layout oder im Listenlayout ausgewählt ist. Das Dialogfeld "Stil" öffnet sich auch dann, wenn ein neues Element hinzugefügt wird. Die folgenden Informationen können in diesem Feld ausgewählt werden :

- ***Grafischer Stil und Einstellungen***

  Der Anzeigestil (einfacher Text, Histogramm, Kurve…) eines Elements kann dynamisch verändert werden. Wenn Vordergrund- und Hintergrundfarben benutzt werden, können sie mit Hilfe der verfügbaren Farbkästchen benutzerspezifisch gestaltet werden. Wenn der Stil "Boolesches Symbol" ausgewählt wurde, muß der Pfadname der entsprechenden .ICO Dateien eingegeben werden. Benutzen Sie die Schaltflächen "**…**" , um die auf der Festplatte vorhandenen Symboldateien zu durchlaufen.

- ***Maßstab***

  Dies ist der Höchstwert, der in einem Histogramm oder in einer Kurve angezeigt werden kann. Bei bipolaren Histogrammen und Kurven gilt dieser Wert sowohl für die positive als auch für die negative Richtung oder Achse.

- ***Variablenname***

  Wenn das Eingabefeld "**Name**" aktiv ist, ermöglicht die Schaltfläche "**…**" die Namen der Variablen anzuzeigen, die bereits im Datenverzeichnis des Projekts deklariert wurden.

- ***Titel***

  Es besteht die Möglichkeit, einen Titel im grafischen Layout neben dem grafischen Element anzuzeigen. Dabei kann der Benutzer die Lage dieses Textfelds (oben, unten, links oder rechts) sowie seinen Inhalt frei bestimmen. Ein Titel kann eine beliebige Kombination des Variablennamens und ihres Werts in Textform sein. Die Benutzerspezifizierung des Titels nimmt keinen Einfluß auf das Listenlayout.

- ***Variable modifizieren***

  Wenn die Option "Variable modifizieren" aktiviert wird, kann der Benutzer beim Debugging den Wert einer Variablen durch Doppelklicken auf das zugeordnete grafische Symbol verändern.

A.18.4. Befehle des "Datei"-Menüs

Mit den Befehlen des "**Datei**"-Menüs kann der Benutzer das komplette Dokument verwalten.

Der Befehl "**Neu**" im "**Datei**"-Menü startet die Bearbeitung eines neuen Dokuments. Es können beliebig viele Dokumente für ein ISaGRAF Projekt definiert werden. Bevor ein neues Diagramm bearbeitet wird, schließt sich das zuvor geöffnete Diagramm. SpotLight kann nicht benutzt werden, um mehrere Diagramme gleichzeitig zu bearbeiten. Dagegen ist es möglich, mehrere SpotLight-Fenster zu öffnen, in denen dann verschiedene Dokumente gleichzeitig bearbeitet werden können.

Der Befehl "**Öffnen**" im "**Datei**"-Menü ermöglicht, das in Arbeit befindliche Dokument zu schließen und die Bearbeitung eines anderen Dokuments aufzunehmen. Das neu geöffnete Dokument ersetzt das aktuelle Dokument im Bearbeitungsfenster. Bei der Auswahl eines neuen Dokuments kann die Schaltfläche "**Löschen**" benutzt werden, um ungewünschte Dateien aus dem Projektverzeichnis zu entfernen. Die in einem Diagramm referenzierten Symbol- und Bitmap-Dateien gehen beim Löschen eines Diagramms nicht verloren.

Der Befehl "**Sichern**" im "**Datei**"-Menü speichert das in Arbeit befindliche Dokument auf der Festplatte. Wenn das Dokument noch keinen Namen besitzt, muß der Benutzer vor der Sicherung einen Namen eingeben. Bei der Benennung eines Dokuments müssen die folgenden Regeln beachtet werden:

- Der Name muß sich auf **8** Zeichen begrenzen.
- Das erste Zeichen muß ein Buchstabe sein.
- Die weiteren Zeichen müssen Buchstaben, Zahlen oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Der Befehl "**Sichern unter**" im "**Datei**"-Menü ermöglicht, das bearbeitete Dokument unter einem anderen Namen zu sichern

A.18.5. Hinweise für ISaGRAF V3.2 Benutzer

Spotlight kann Grafiken und Steuerungsdiagrammlisten lesen, die mit den Werkzeugen von ISaGRAF V3.0 oder V3.2 erstellt wurden. Solche Dateien erscheinen im Dialogfeld "**Öffnen**" mit ihrer Ursprungsbeschreibung und können mit SpotLight gelesen und frei modifiziert werden.
Wenn eine Grafik der Version ISaGRAF V3.2 geöffnet wird, erscheint sie automatisch als "verriegelt". Deaktivieren Sie die Option "Verriegeln" im "**Datei**"-Menü, wenn Sie die Grafik modifizieren möchten.
Wenn eine Grafik oder eine Steuerungsdiagrammliste der Version ISaGRAF 3.2 geöffnet wird, schlägt StopLight vor, sie im nativen SpotLight-Format zu sichern. Das Dialogfeld "**Sichern unter**" öffnet sich systematisch, wenn ein solches Dokument geschlossen wird.

A.19. Anwendungen rücklesen

ISaGRAF unterstützt das Rücklesen der im Zielsystem gespeicherten Anwendung. Das Rückleseverfahren kommuniziert mit dem Zielsystem, um den eingebetteten komprimierten Quellcode (embedded zipped source code - EZS) zu laden und das geladene Projekt in der Workstation-Umgebung wiederherzustellen.
Das im angeschlossenen Zielsystem laufende Projekt kann rückgelesen werden, wenn die Version des Zielsystems V3.22 (oder höher) ist und wenn komprimierter Quellcode mit der Anwendung eingebettet wurde. Das Einbetten von Quellcode ist optionell.

A.19.1. Projekt rücklesen

Das Dialogfeld "**Rücklesen**" wird mit dem Befehl "**Datei**" im ISaGRAF Projektmanager geöffnet. Das Rücklesen bezieht sich nicht auf ein bestehendes Projekt der Workstation. Das in der Projektliste ausgewählte aktuelle Projekt steht in keinem Zusammenhang mit dem Rückleseverfahren. Um die im Zielsystem laufende Anwendung rückzulesen, sind die folgenden Aktionen erforderlich :
1- Prüfen Sie den Anschluß des Zielsystems.
2- Stellen Sie die Kommunikationsparameter entsprechend der Anschlußverbindung ein.
3- Drücken Sie die Schaltfläche "**Run**".

Das Rücklesen des komprimierten Quellcodes (embedded zipped source - EZS) und die Dekomprimierung kann einige Sekunden dauern. Meldungen im Dialogfeld zeigen an, wenn das Rücklesen beendet ist oder ob Fehler aufgetreten sind.

Für die Erstellung des ISaGRAF Projekts wird der Name benutzt, der bei der Kommunikation im Zielsystem gelesen wurde. Wenn ein bestehendes Projekt der Workstation diesen Namen bereits besitzt, werden Sie aufgefordert, das Projekt zu überschreiben oder einen unbenutzten Namen zu wählen. Nach dem Rücklesen kann die Speicherung des geladenen Quellcodes als Projekt nicht mehr widerrufen werden. Das rückgelesene Projekt kann jetzt geöffnet werden.

Mögliche Fehler

Die folgenden Fehler können beim Rücklesen eines Projekts auftreten. Die Fehlermeldungen erscheinen im Dialogfeld "Rücklesen".
- Kommunikation mislungen.
- Das Rücklesen wird nicht von ISaGRAF Zielsystemen vor Version 3.22 unterstützt.
- Keine Anwendung im Zielsystem.
- Kein Quellcode verfügbar oder Rücklesen unmöglich.

A.19.2. Kommunikationsverbindung

Drücken der Schaltfläche "**Einstellung**" ermöglicht die Definition der Verbindungsparameter für die Rücklese-Kommunikation der ISaGRAF Workstation mit dem ISaGRAF Zielsystem. Vergewissern Sie sich, daß die konfigurierten

Parameter dem angeschlossenen Zielsystem entsprechen, bevor Sie das Rücklesen einleiten.

A.19.3. Projekt zum Rücklesen vorbereiten

Zunächst wird der ISaGRAF Code-Generator angewiesen, den komprimierten Quellcode zusammen mit dem Anwendungscode einzubetten, um ein späteres Rücklesen zu ermöglichen. Drücken Sie die Schaltfläche "**Rücklesen**"im Dialogfeld "**Kompilierungsoptionen**". In einem weiteren Dialogfeld kann das Einbetten des komprimierten Quellcodes als Option aktiviert werden. In diesem Fall wird nur ein Minimum an unbedingt erforderlichen Quelldateien eingebettet. Wenn darüber hinaus noch zusätzliche, optionelle Dateien eingebettet werden sollen, können weitere Optionen aktiviert werden.

Wichtiger Hinweis: Die Bibliotheken werden nicht mit dem eingebetteten Quellcode geladen. Hierzu gehören die Funktionen, Funktionsbausteine, E/A-Karten und Baugruppen.

Optionale Dateien

Zusätzlich zum unbedingt erforderlichen Quellcode können wahlweise die folgenden Dateien eingebettet werden. Ihre Einbettung erfordert zusätzlichen Speicherplatz im Zielsystem.

- Projektkopf
Wenn der Projektkopf nicht eingebettet wird, zeigt er nach dem Rücklesen lediglich das Rücklesedatum an.

- Paßwortschutz
Die Rücklesefunktion ist nicht paßwortgeschützt. Wenn das rückgelesene Projekt geschützt werden soll, muß sein Paßwortschutzsystem mit dem Quellcode eingebettet werden.

- Kommentare für nicht angeschlossene E/A-Kanäle
ISaGRAF gibt Ihnen die Möglichkeit, eine Textbeschreibung für nicht angeschlossene E/A-Kanäle einzugeben. Diese Option darf nicht aktiviert werden, wenn nur mit angeschlossenen E/As gearbeitet wird.

- Projektprotokoll
Dies ist das globale Änderungsprotokoll eines Projekts.

- Programmprotokolle
Die Protokolldatei eines Programms enthält Benutzernotizen sowie das Protokoll der Compiler-Ausgangsmeldungen bezüglich des Programms. Das Einbetten von Programmprotokolldateien kann sehr viel Speicherplatz im Zielsystem in Anspruch nehmen.

- Variablenlisten und Steuerungsdiagramme
Diese Datein werden beim Debugging erstellt und enthalten die Listen der Variablennamen für die Anzeige von Listen und Steuerungsdiagrammen.

- Grafiken, Symbole und Bitmaps

Dies sind die ISaGRAF Grafiken sowie alle zugeordneten Symbol- und Bitmapdateien, wenn sie sich im Projektverzeichnis befinden. Warnung: Das Einbetten von Grafiken kann sehr viel Speicherplatz im Zielsystem beanspruchen.

A.19.4. Wie komprimierter Quellcode im Zielsystem gespeichert wird

Der eingebettete komprimierte Quellcode (embedded zipped source - EZS) wird im generierten Code mit den Ressourcen gespeichert. Die generierte Ressource heißt "**EZS**". Wenn das Einbetten von Quellcode ausgewählt wurde, kann dieser Name für keine andere Ressource benutzt werden. Durch das Einbetten von Quellcode wird weder die Definition von Ressourcen begrenzt, noch wird die Definitionsdatei der Benutzer-Ressourcen beeinflußt.
Siehe ISaGRAF Dokumentation zum Thema "Code-Entwicklung" für zusätzliche Informationen über die Ressourcen.

A.19.5. Speicheranforderungen des Zielsystems

Eingebetteter komprimierter Quellcode (embedded zipped source - EZS) benötigt zusätzlichen Speicherplatz, um mit dem Anwendungscode im Zielsystem gespeichert werden zu können. Schätzungsweise ist der EZS (ohne zusätzliche Optionen) eineinhalbmal so groß wie der ausführbare Code. Folglich ist der insgesamt geladene Code durch das Einbetten des EZS ca. 2,5 mal so umfangreich.
Bei manchen Zielsystemen mit segmentiertem Speicher gibt es bestimmte Begrenzungen. Da der EZS als Ressource im generierten Code gespeichert wird, muß er im gleichen Datensegment wie der Anwendungcode gespeichert werden.

A.19.6. Informationen zum rückgelesenen Projekt

Das rückgelesene Projekt enthält sämtliche für eine Neukompilierung erforderlichen Dateien und Daten. Abhängig von den Optionen, die bei der vorherigen Kompilierung ausgewählt wurden, kann es Zusatzdateien enthalten, beipielsweise Projektkopf- und Programmprotokolldateien.
Das Projekt muß kompiliert werden, bevor man es debuggen oder anzeigen kann. Warnung: Da ISaGRAF einen Codekompilierungsstempel benutzt, um Anwendungen miteinander zu vergleichen, wird der Benutzer beim Öffnen des Debuggers informiert, daß die Workstation- und Zielsystemanwendungen unterschiedliche Versionscodes besitzen.

Wichtiger Hinweis: Die Bibliotheken werden nicht mit dem eingebetteten Quellcode rückgelesen. Vergewissern Sie sich, daß die betreffenden Bibliotheksfunktionen und -funktionsbausteine in Ihrer ISaGRAF Workstation installiert sind, bevor Sie die rückgelesene Anwendung neu kompilieren.

A.19.7. Kompatibilität

Durch Erweiterungen im Kommunikationsprotokoll wird das Rückleseverfahren von ISaGRAF Zielsystemen und der Workstation Version 3.22 oder höher unterstützt.

Es besteht die Möglichkeit, komprimierten Quellcode (EZS) in ein Zielsystem einzubetten, das auf den ISaGRAF Systemen Version 3.03 bis 3.21 basiert, da der EZS im Anwendungscode als Standard-Ressource gespeichert wird. Hingegen ist es bei Systemen dieser Art nicht möglich, die eingebetteten Daten rückzulesen, da sie die erforderlichen Kommunikation-Services nicht unterstützen.

A.20. Diagnosewerkzeug benutzen

Das "**Diagnosewerkzeug**" ist ein untergeordneter Teil des ISaGRAF Debuggers. Es gibt dem Endbenutzer die Möglichkeit, mit einer vorbestimmten Variablengruppe zu arbeiten, um den Prozeß zu überwachen und zu steuern. Der ISaGRAF Debugger selbst ist ein äußerst wirkungsvolles Werkzeug, das Funktionen hohen Niveaus einschließt. Mit dem Diagnosewerkzeug können Zielanwendungen in den Betriebs- und Instandhaltungsphasen sicher und gefahrlos gesteuert werden. Das ISaGRAF Diagnosewerkzeug wird direkt in der ISaGRAF Gruppe im Programm-Manager durch Doppelklicken auf das folgende Symbol gestartet:

Die Liste der bestehenden Projekte erscheint in einem Dialogfeld. Hier kann der Endbenutzer den begrenzten ISaGRAF Debugger für eine bestehende und schon geladene ISaGRAF Anwendung starten. Klicken auf "**OK**" startet den begrenzten Debugger für das ausgewählte Projekt. Klicken auf die Schaltfläche "**Abbrechen**" schließt das Dialogfeld. Der Befehl "**Konfigurieren**" wird benutzt, um die Kommunikationsverbindung zwischen der ISaGRAF Workstation und der Ziel-SPS herzustellen. Siehe Kapitel "**Programmverwaltung**" in diesem Handbuch für weitere Informationen zu diesem Befehl.

Hinweis: Das ISaGRAF Diagnosewerkzeug (begrenzter Debugger) kann nicht benutzt werden, um die in der Ziel-SPS betriebene Anwendung zu laden, zu stoppen oder zu aktualisieren. Es können keine Operationen ausgeführt werden, wenn das im Dialogfeld "Diagnosewerkzeug" ausgewählte Projekt nicht mit dem Projekt übereinstimmt, welches in der SPS installiert und aktiv ist.

Wenn der begrenzte ISaGRAF Debugger gestartet wird und ordnungsgemäß an die Zielanwendung angeschlossen ist, sind die folgenden Befehle verfügbar:

- Variablenlisten überwachen
- Grafische Dokumente mit SpotLight überwachen

A.21. ISaGRAF Simulator benutzen

Der ISaGRAF Kernelsimulator wird mit dem Debugger gestartet, wenn der Befehl "Simulieren" im Menü "Debuggen", im Fenster des Programmanagers, eingeleitet wird. Der Kernelsimulator ist ein komplettes ISaGRAF Zielsystem, das die ISaGRAF Standardfunktionalitäten und sämtliche "C"-Funktionen und -Funktionsbausteine der von CJ International gelieferten Standardbibliothek unterstützt. Die E/A-Karten werden grafisch in einem Fenster simuliert. Alle E/A-Kartentypen können simuliert werden. Karten, welche bei der E/A-Verdrahtung als "Virtuelle Karten" definiert wurden, erscheinen ebenfalls im Simulatorfenster.

A.21.1. Verknüpfungen mit dem Debugger

Der Kernelsimulator unterstützt die uneingeschränkte Kommunikation mit dem ISaGRAF Debugger, so daß sämtliche Debugging-Funktionen bei der Simulation benutzt werden können. Der Kernelsimulator arbeitet stets mit der aktuellen ISaGRAF Anwendung. In der Simulationsphase sind die Debugger-Befehle "**Starten**", "**Stoppen**", "**Übertragen**" und "**Aktualisieren**" nicht mehr verfügbar. Der Simulator kann nur benutzt werden, wenn die Compiler-Option "**SIMULATE**" vor der Entwicklung des Zielcodes ausgewählt wurde. Wenn Sie das Simulatorfenster schließen, schließt sich auch das Debugger-Fenster (und alle anderen während der Debugging-Sitzung geöffneten Fenster).

A.21.2. E/A-Simulation

Die E/A-Karten erscheinen im Simulatorfenster mit ihren Namen und Steckplatznummern. Sämtliche ISaGRAF E/A-Standardtypen (boolesch, analog und Zeichenkette) können simuliert werden. Die Kanäle der Eingangskarten werden durch spezielle Schaltflächen und Felder, die Kanäle der Ausgangskarten durch grafische Lichtanzeigen und Datenfelder dargestellt.

Boolesche Eingänge: Ein boolescher Eingang wird durch eine rechteckige, grüne Schaltfläche dargestellt. Die Kanalnummer erscheint neben der E/A-Schaltfläche. Der Eingangswert ist TRUE, wenn die Schaltfläche "gedrückt" ist. Klicken auf die Schaltfläche invertiert den Zustand des entsprechenden Eingangs. Wenn die Schaltfläche gedrückt ist, benutzen Sie die rechte Maustaste, um den Eingang zu forcieren.

Boolesche Ausgänge: Ein boolescher Ausgang wird durch einen kleinen Kreis dargestellt. Die Kanalnummer erscheint neben der E/A-Schaltfläche. Der Ausgangswert ist TRUE, wenn sich das grafische Symbol farblich abhebt.

Analoge Eingänge: Ein analoger Eingangskanal ist ein einfaches, numerisches Feld, in dem der Wert des entsprechenden Eingangs eingegeben werden kann. Klicken auf das Feld aktiviert den Eingabecursor. Jetzt kann der neue Wert für den Kanal eingegeben werden. Bestätigen mit der **EINGABE**-Taste ist überflüssig. Analoge Eingänge können als Dezimal- oder Hexadezimalwerte eingegeben

werden. Benutzen Sie die Schaltflächen "Nach oben/unten", um den aktuellen Wert zu vergrößern oder zu verkleinern.

Analoge Ausgänge: Ein analoger Ausgangskanal wird als numerisches Ausgangsfeld dargestellt. Der Ausgangswert kann als Dezimal- oder Hexadezimalzahl angezeigt werden. Ausgangskanäle können nicht vom Anwender geändert werden.

Zeichenketten-Eingang: Ein Eingangskanal vom Typ Zeichenkette ist ein einfaches Textfeld, in dem der Wert des entsprechenden Eingangs eingegeben wird. Klicken auf das Feld aktiviert den Eingabecursor. Jetzt kann der neue Wert für den Kanal eingegeben werden. Bestätigen mit der **EINGABE**-Taste ist überflüssig.

Zeichenketten-Ausgang: Ein Ausgangskanal vom Typ Zeichenkette ist ein Textfeld. Ausgangskanäle können nicht vom Anwender geändert werden.

A.21.3. Bibliothekskomponenten

Der ISaGRAF Simulator unterstützt die von CJ International gelieferten Standardumrechnungen, -funktionen und -funktionsbausteine. Die folgenden Objekte werden unterstützt:

- ***Umrechnungsfunktionen:***

  bcd, scale

- ***Funktionen:***

  abs, acos, ArCreate, ArRead, ArWrite, ascii, asin, atan, char, cos, delete, expt, find, insert, left, limit, log, max, mid, min, mlen, mod, mux4, mux8, odd, rand, replace, right, rol, ror, sel, shl, shr, sin, sqrt, tan, trunc

- ***Funktionsbausteine:***

  average, blink, cmp, ctd, ctu, ctud, derivate, f_trig, hyster, integral, lim_alrm, r_trig, rs, sema, sr, stackint, tof, ton, tp

Benutzerdefinierte Umrechnungen, "C"-Funktionen und -Funktionsbausteine werden gewöhnlich nicht vom ISaGRAF Simulator integriert. Solche Objekte werden entwickelt, um Software- und Harwareressourcen des Zielsystems zu nutzen. Ressourcen dieser Art sind generell nicht im Windows-System verfügbar. Der ISaGRAF Simulator bewirkt das folgende Standardverhalten bei allen benutzerdefinierten Umrechnungen, Funktionen und Funktionsbausteinen:

- Wenn eine neue Umrechnung vom Simulator verarbeitet wird, wird sie durch eine "Null"-Umrechnung ersetzt. Dies bedeutet, daß der physikalische Wert analoger Variablen immer gleich dem elektrischen Wert ist (wie in der Simulatortafel eingegeben oder angezeigt).
- Wenn neue "C"-Funktionen oder -Funktionsbausteine vom Simulator verarbeitet werden, erfolgt keine Operation. Der Rückgabewert ist gleich null.

A.21.4. Optionen

Mit den Befehlen im Menü "**Optionen**" kann die Bildschirmanzeige der E/A-Elemente in der Simulatortafel variiert werden. Diese Optionen können vom Benutzer während der Debugging-Sitzung jederzeit aktiviert oder deaktiviert werden.

- Wenn die Option "**Farbanzeige**" aktiviert wird, erscheinen die E/A-Kanäle als farbige Bitmaps. Falls sich die Farben auf LCD-Bildschirmen nicht deutlich genug abheben, sollte man diese Option deaktivieren. Die Ein- und Ausgänge der E/A-Kanäle erscheinen dann in schwarz/weiß.
- Wenn die Option "**Variablennamen**" aktiviert wird, erscheint ein Etikett mit dem Namen der angeschlossenen E/A-Variablen neben dem entsprechenden E/A-Kanal. Durch Deaktivieren dieser Option kann die Simulatortafel verkleinert werden.
- Wenn die Option "**Hexadezimalwerte**" aktiviert wird, erfolgt die Eingabe und Anzeige aller analogen Ein- und Ausgangskanäle im Hexadezimalformat.
- Wenn die Option "**Immer sichtbar**" aktiviert wird, bleibt das Simulatorfenster immer sichtbar, selbst wenn Eingaben in anderen Fenstern erfolgen.

A.21.5. Eingangszustände sichern und wiederherstellen

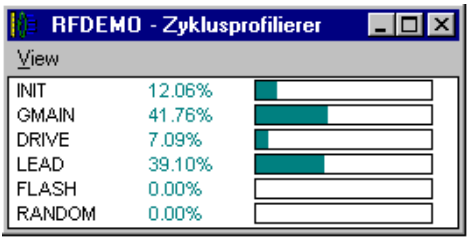
Unter Anwendung des ISaGRAF Simulators werden Eingangskanäle manuell forciert (Schaltflächen und Textfelder der Simulatortafel). Die folgenden Befehle des "**Werkzeug**"-Menüs können jederzeit benutzt werden, um die Zustände aller Eingangskanäle zu sichern und wiederherzustellen:

Eingangsschema laden: Forcieren der Eingangskanäle mit den Werten, die auf der Festplatte in einer Datei durch den Befehl "Eingangsschema sichern" gespeichert wurden.
Eingangsschema sichern: Sichern der Eingangskanäle in einer Datei, so daß sie zu einem späteren Zeitpunkt mit dem Befehl "Eingangsschema laden" wiederhergestellt werden können. Die Datei wird im Projektverzeichnis gespeichert, also zusammen mit den anderen Projektdateien von der ISaGRAF Archivierungs-Utility gesichert.

Hinweis: Nur benannte Eingangskanäle (d.h. Kanäle, die mit einer Variablen verbunden sind) werden auf Platte gesichert.

A.21.6. Der Zyklusprofilierer

Der ISaGRAF Zyklusprofilierer ist ein leistungsfähiges Diagnosewerkzeug, das errechnet, wieviel Zykluszeit die einzelnen Programme, Funktionen und Funktionsbausteine einer Anwendung in Anspruch nehmen. Dieses Werkzeug bietet sich an, wenn eine schnelle Diagnose der Leistungen einer Anwendung erforderlich ist und gibt dem Programmierer die Möglichkeit, auf Codeabschnitte zuzugreifen, die eventuell optimiert werden könnten.
Der Zyklusprofilierer wird mit dem Befehl "**Werkzeuge / Zyklusprofilierer**" in den Menüs des ISaGRAF Simulatorfensters eingeleitet. Er zeigt für jedes Programm, jede Funktion und jeden Funktionsbaustein an, wieviel Zykluszeit (in Prozent) deren Ausführung in Anspruch nimmt:



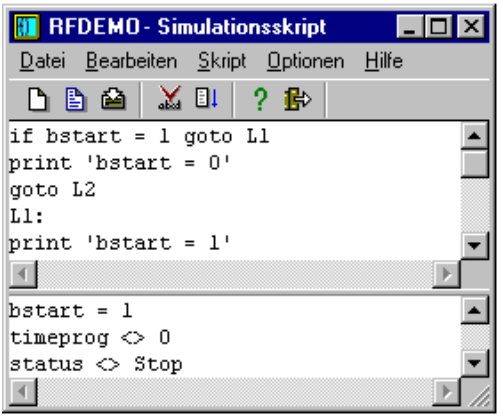
Wenn die Option "**Anzeige / Durchschnitt**" aktiviert wird, erscheint der Durchschnitt der errechneten Prozente seit dem Start der Anwendung (oder seit dem letzten Befehl "**Anzeige / Rücksetzen**").
Wenn die Option "**Anzeige / Durchschnitt**" nicht aktiviert wird, erscheinen die Messungen, die während der Ausführung des letzten Zyklus erfolgt sind. Diese Funktion kann auch benutzt werden, wenn sich die Anwendung im "**Schrittmodus**" befindet, um Messungen im Zusammenhang mit dem Anwendungskontext zu erhalten.
Der Befehl "**Anzeige / Kopieren**" wird benutzt, um die Programmnamen mit den errechneten Prozenten im ASCII-Format in die Windows-Zwischenablage zu kopieren. Diese Daten können dann in Textdokumente oder herkömmliche SpreadSheets eingefügt werden.

Wichtige Hinweise:

Diese Messungen sind nicht exakt. Die Prozentsätze werden durch Zählen der TIC-Anweisungen berechnet, wobei die verschiedenen Ausführungszeiten der Anweisungen berücksichtigt werden. Die Berechnungen beinhalten nicht die Zeit, die in "C"-Funktionen und -Funktionsbausteinen verbraucht wird.
Der für eine Funktion oder einen Funktionsbaustein angezeigte Wert ist die Summe aller "Aufrufzeiten" der Anwendungsprogramme im selben Zyklus.
Die Zeitberechnung stützt sich auf TIC-Code und liefert keine zuverlässigen Daten, wenn der aktuelle Anwendungscode in der Hochsprache "C" generiert und mit einem "C"-Compiler kompiliert wurde.

A.21.7. Simulationsskript

Der ISaGRAF Simulator verfügt über ein Werkzeug, mit dem Simulationsskript erstellt und ausgeführt werden kann. Das Skript wird in einer einfachen, ST-ähnlichen Textsprache geschrieben und dient der Automatisierung von ISaGRAF Simulatortests.
Der Simulatiosskripteditor wird mit dem Befehl "**Werkzeuge / Simulationsskript**" im Simulatorfenster gestartet. Es folgt das Fenster des Skripteditors:



Das obere Fenster ist ein Texteditor, in dem Skriptanweisungen eingegeben werden. Er wird wie die anderen ISaGRAF Texteditoren benutzt und verfügt über hochgradige Funktionen, wie die Auswahl von Variablensymbolen mit der Maus. Die Befehle des Menüs "**Optionen**" werden benutzt, um Tabulationen zu setzen und eine Schriftart auszuwählen.
Im unteren Fenster erscheinen die entsprechenden Ausgabemeldungen, wenn das Skript läuft. Die Trennlinie zwischen den beiden Fenstern kann beliebig verschoben werden. Das Ausgabefenster kann bei der Skripbearbeitung geschlossen werden, öffnet sich aber jedesmal automatisch, wenn ein Skript ausgeführt wird.

Skript bearbeiten

Benutzen Sie die Befehle des "Datei"-Menüs, um die Skriptdateien zu manipulieren:

Neu	erstellt ein neues, unbenanntes Skript
Öffnen	lädt ein bestehendes Skript aus der Datei
Sichern	sichert Skripttext und Inhalt des Ausgabefensters auf der Festplatte (im Projektverzeichnis)
Sichern unter	sichert das Skript unter einem anderen Namen

Für jedes Skript werden im ISaGRAF Projektverzeichnis zwei Dateien erstellt:
<*skriptname*>.SCC Text des Skripts (Anweisungen)
<*skriptname*>.SCO Inhalt des Ausgabefensters
wobei <*skriptname*> der Name des Skripts ist.

Beide Dateien sind Standardtextdateien und können mit einem beliebigen Texteditor geöffnet werden.
Bei der Bearbeitung eines Skripts wird der Befehl "**Bearbeiten / Symbol einfügen**" benutzt, um einen deklarierten Variablennamen auszuwählen und am Standort des Eingabecursors einzufügen.

Skript ausführen

Ein Skript muß geprüft und kompiliert werden, bevor es ausgeführt werden kann. Falls erforderlich, erfolgt die Syntaxprüfung bei einem "Run"-Befehl automatisch. Benutzen Sie die folgenden Befehle des Menüs "**Skript**":

Prüfen prüft die Syntax und kompiliert das Skript

Skript ausführen startet die Ausführung des bearbeiteten Skripts

Bevor ein neues, unbenanntes Skript geprüft werden kann, muß es mit einem Namen gesichert werden. Der Inhalt eines benannten Skripts wird vor der Syntaxprüfung automatisch auf der Festplatte gesichert.
Es ist nicht möglich, den Inhalt eines Skripts während der Ausführung zu modifizieren. Wenn das Ende des Skripts erreicht ist, erscheint eine entsprechende Meldung im Ausgabefenster. Mit dem folgenden Befehl im Menü "**Skript**" kann die Ausführung eines Skripts abgebrochen werden:

Skript abbrechen beendet ein in der Ausführung befindliches Skript

Die Skriptausführung erfolgt zwischen den Zielsystemzyklen. Falls eine unendliche Schleife im Zyklus programmiert ist, wird diese vom ISaGRAF Simulator unterbrochen, so daß die ISaGRAF Zyklen weiterhin ausgeführt und andere ISaGRAF Anwendung nicht blockiert werden. Wenn die gleiche Marke mehrmals in einem Zielsystemzyklus angetroffen wird, wird das Skript vom ISaGRAF Skriptinterpretierer unterbrochen. Die Skriptausführung kann auch auf normalem Weg durch "Cycle"- oder "Wait"-Anweisungen unterbrochen werden.

Skriptsprache

Die Sprache für die Beschreibung von Skript ist eine einfache, ST-ähnliche Textsprache, in der jede Anweisung auf einer separaten Textzeile eingegeben wird, die am Zeilenende kein Semikolon benötigt. Die folgende Schaltfläche in der Toolbox dient der Anzeige der verfügbaren Anweisungen und der Auswahl eines Schlüsselworts, das dann am Standort des Eingabecursors eingefügt wird:

Anweisung einfügen (Schlüsselwort und Hilfe als Kommentar)

Es gibt verschiedene Anweisungstypen. Der erste Typ ist die Zuweisung (Forcierung) einer Variablen:

:= Assignment

Die folgenden Anweisungen bewirken die Ausgabe von Meldungen im Ausgabefenster:

Print Augabe einer Zeichenkette oder eines Variablenwerts
PrintTime Ausgabe von aktuellem Datum und Uhrzeit

Die folgenden Anweisungen werden benutzt, um Skriptanweisungen mit dem ISaGRAF Zyklus zu synchronisieren:

Cycle läßt den ISaGRAF Simulator einen Zyklus ausführen
Wait wartet auf eine Verzögerung

Weitere Anweisungen steuern den Datenfluß der Anweisungen im Skript:

Labels	können an beliebigen Stellen im Skript plaziert werden
Goto	unbedingter Sprung zu einer Marke (label)
If goto	bedingter Sprung zu einer Marke (label)
End	beendet das Skript

Skript kann wahlweise groß- oder kleingeschrieben werden. Am Ende der Textzeilen können Kommentare eingefügt werden, die entweder gemäß der ST-Konventionen (zwischen "**(***" und "***)**" Zeichen) geschrieben oder mit dem Vorzeichen "**;**" behaftet werden.

":=" Assignment

Bedeutung: Forciert den Wert einer ISaGRAF Variablen, die eine interne Variable oder ein Ein- oder Ausgangskanal sein kann.

Syntax: <varname> **:=** <konstanter_ausdruck>
<varname> **=** <konstanter_ausdruck>

Argumente: <varname> ist das gültige Symbol einer deklarierten Anwendungsvariablen oder eine direkt (gemäß der "%" Schreibkonventionen) dargestellte E/A-Variable.
<konstanter_ausdruck> ist ein gültiger konstanter Ausdruck, der dem Typ der spezifizierten Variablen entspricht. Bei booleschen Variablen kann "0" und "1" anstatt "FALSE" und "TRUE" benutzt werden. Bei Timern kann das Vorzeichen "T#" oder "TIME#" weggelassen werden.

Hinweise: Eingangsvariablen, die von einem Skript forciert werden, brauchen nicht verriegelt zu sein. Die Zeichnung des entsprechenden Eingangskanals wird aktualisiert, wenn eine Eingangsvariable von einem Skript forciert wird.
Warnung: Analoge Ein- oder Ausgangsvariablen, die mit einer Umrechnung verbunden sind, dürfen nicht forciert werden, da die Skriptausführung keine Umrechnungsfunktionen oder -tabellen unterstützt.

Beispiel:
```
MyBooVar := 1          (* äquivalent zu TRUE *)
MyIntVar := 1234
MyRealVar := 1.2345
MyMsgVar := 'Hello'
```

Print

Bedeutung: Schreibt eine Zeichenkette oder einen Variablenwert im Ausgabefenster. Der Text erscheint als neue Zeile am Ende des bereits im Ausgabefenster angezeigten Texts.

Syntax: **Print '***<text>***'**

Print <*varname*>

Argumente: <text> ist eine beliebige, in einfachen Anführungszeichen geschriebene Zeichenkette
<varname> ist das gültige Symbol einer deklarierten Anwendungsvariablen oder eine direkt (gemäß der "%" Schreibkonventionen) dargestellte E/A-Variable.

Hinweis: Das Ausgabeformat von Variablenwerten entspricht stets den IEC-Konventionen.

Beispiel:
```
Print 'Hello'
Print MyBooVar
```

Ausgabe:
```
Hello
MyBoovar = TRUE
```

PrintTime

Bedeutung: Schreibt das aktuelle Datum und die Uhrzeit als neue Zeile an das Ende des im Ausgabefenster angezeigten Texts.

Syntax: **PrintTime**

Hinweis: Datum und Uhrzeit erscheinen im aktuellen Format des Windows-Systems.

Beispiel:
```
Print 'Time now is:'
PrintTime
```

Ausgabe:
```
Time now is:
15:45:22
```

Cycle

Bedeutung: Hält die Skriptausführung bis zum nächsten ISaGRAF Zyklus zurück.

Syntax: **Cycle**

Hinweis: Skriptanweisungen werden am Anfang eines ISaGRAF Zyklus ausgeführt. Wenn sich der Simulator im "Schrittmodus" befindet, folgt ein Zyklus unmittelbar auf die Anweisung "cycle". Die nachfolgenden Skriptanweisungen erfolgen beim nächsten Befehl "Einen Zyklus ausführen" des Debuggers.

Beispiel:
```
(* ISaGRAF Programm kopiert A zu B *)
A := 0
Cycle
Print B (* noch kein Zyklus ausgeführt *)
```

```
A := 1
Print B (* kein Zyklus ausgeführt *)
        (* B nicht auf 1 gesetzt *)
Cycle
Print B
```

Ausgabe:
```
B = 0
B = 0
B = 1
```

Wait

Bedeutung: Hält die Skriptausführung zurück, bis die Verzögerungszeit abgelaufen ist.

Syntax: **Wait** <verzögerung>

Argumente: <verzögerung> wird gemäß der IEC-Konventionen für konstante Zeitausdrücke geschrieben. Die Vorzeichen "T#" oder "TIME#" können weggelassen werden. Die Verzögerung muß zwischen 10 Millisekunden und 1 Stunde liegen.

Hinweis: Die "Wait"-Anweisung ist mehr oder weniger präzise und abhängig vom Windows-System. Zudem sollte man bei einer Verzögerung plus/minus einen ISaGRAF Zyklus berücksichtigen.

Beispiel:
```
PrintTime
Wait 2s
PrintTime
```

Ausgabe:
```
15:45:27
15:45:29
```

Labels

Bedeutung: Marken können an beliebigen Stellen im Zyklus eingesetzt werden. Sie werden als Ziel für die "Goto"-Anweisungen benutzt und ermöglichen die Datenflußsteuerung von Skriptanweisungen.

Syntax: <markenname>**:**

Argumente: <markenname> ist ein einmaliger Name, der gemäß der IEC-Konventionen für die Variablenbenennung geschrieben wird: Der Name begrenzt sich auf 16 Zeichen, beginnt mit einem Buchstaben, auf den Buchstaben, Zahlen oder Unterstrichzeichen folgen. Auf den Markennamen sollte ein "**:**" Zeichen folgen.

Hinweis: Eine Zeile, auf der eine Marke definiert ist, darf keine Anweisungen enthalten.

Markennamen sollten nicht für deklarierte ISaGRAF Variablen benutzt werden.

Beispiel:
```
(* Skripts mit unbegrenzter Schleife *)
loop:
PrintTime
Wait 1s
Goto loop
```

Goto

Bedeutung: Unbedingter Sprung zu einer Marke

Syntax: **Goto** <markenname>

Argumente: <markenname> ist der Name einer im Skript definierten Marke.

Hinweis: Rückwärtssprünge sind erlaubt. Bei unbegrenzten Schleifen wird die Skriptausführung in jeder Schleife automatisch unterbrochen, um die Ausführung der ISaGRAF Zyklen zu gewährleisten.

Beispiel:
```
Print 'Before Jump'
Goto MyLabel
Print 'In Jump' (* nie ausgeführte Anweisung *)
MyLabel:
Print 'After Jump'
```

Ausgabe:
```
Before Jump
After Jump
```

If Goto

Bedeutung: Bedingter Sprung zu einer Marke. Die Bedingung ist entweder ein Vergleich zwischen zwei ISaGRAF Variablen oder ein Vergleich zwischen einer Variablen und einem konstanten Ausdruck.

Syntax: **If** <var1> ***test*** <var2> **Goto** <markenname>
If <var1> ***test*** <konstanter_ausdruck> **Goto** <markenname>

Verfügbare Vergleichstests sind:
= true, wenn beide Teile den gleichen Wert haben
<> true, wenn die Teile unterschiedliche Werte haben
< true, wenn erster Teil kleiner als zweiter Teil ist
<= true, wenn erster Teil kleiner oder gleich zweiter Teil ist
> true, wenn erster Teil größer als zweiter Teil ist
>= true, wenn erster Teil größer oder gleich zweiter Teil ist

Argumente: <var1> <var2> sind gültige Symbole von deklarierten Anwendungsvariablen oder direkt, unter Anwendung der "%" Schreibkonventionen, dargestellte E/A-Variablen.

<konstanter_ausdruck> ist ein gültiger konstanter Ausdruck, der dem Typ der spezifizierten Variablen entspricht. Bei booleschen Variablen kann "0" und "1" anstatt "FALSE" und "TRUE" benutzt werden. Bei Timern kann das Vorzeichen "T#" oder "TIME#" weggelassen werden.
<markenname> ist der Name einer im Skript definierten Marke.

Hinweis: Rückwärtssprünge sind erlaubt. Bei unbegrenzten Schleifen wird die Skriptausführung in jeder Schleife automatisch unterbrochen, um die Ausführung der ISaGRAF Zyklen zu gewährleisten.

Beispiel:
```
(* Dieses Skript läuft in Schleife, *)
(* bis MyVar TRUE ist *)
Loop:
If MyVar = FALSE Goto Continue
Goto TheEnd
Continue:
Print MyVar
Goto Loop
TheEnd:
```

End

Bedeutung: Beendet das Skript.

Syntax: **End**

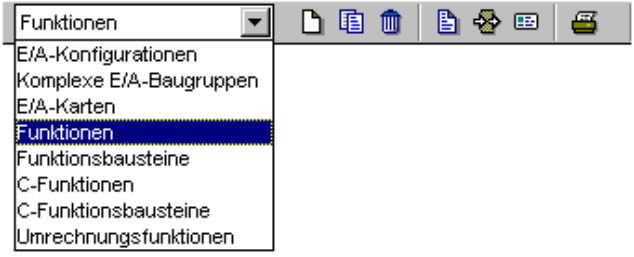
Hinweis: Es ist nicht unbedingt erforderlich, eine "End"-Anweisung auf die letzte Zeile eines Skripts zu setzen.

Beispiel:
```
(* Dieses Skript läuft in Schleife, *)
(* bis MyVar TRUE ist *)
Loop:
If MyVar = FALSE Goto Continue
End
Continue:
Print MyVar
Goto Loop
```

A.22. Bibliotheksmanager benutzen

Die ISaGRAF Bibliotheken bilden die Standard-Schnittstelle zwischen der Automatisierungsentwicklung und den Software- und Hardware-Kapazitäten des ISaGRAF Zielsystems. Für jeden Schnittstellentyp gibt es eine Bibliothek. Der ISaGRAF Bibliotheksmanager ist sowohl für Hardware-Hersteller als auch für Software-Ingenieure bestimmt. Er dient der Beschreibung der ISaGRAF Programmierschnittstelle der entwickelten Objekte.

Der Bibliotheksmanager der ISaGRAF Workstation zeigt jeweils die Elemente einer der ISaGRAF Bibliotheken an. In der linken Fensterhälfte erscheinen die **Elemente** der ausgewählten Bibliothek in einer Liste. Rechts im Fenster erscheint das **technische Datenblatt** (Gebrauchsanleitung) des in der Liste ausgewählten Elements. Die Menüs des Bibliotheksmanagers enthalten Befehle zur Erstellung, Bestimmung und Änderung der Elemente der aktiven Bibliothek. Der Befehl "**Datei / Andere Bibliothek**" ermöglicht die Auswahl einer anderen ISaGRAF Bibliothek. Das Listenfeld links neben der Befehlsleiste kann ebenfalls für die Auswahl einer Bibliothek benutzt werden:



A.22.1. Bibliothekselemente verwalten

Benutzen Sie die Befehle im "**Datei**"-Menü, um Elemente zu erstellen und bestehende Elemente zu bearbeiten.

Ein neues Element erstellen

Der Befehl "**Neu**" im "**Datei**"-Menü erstellt ein neues Element in der ausgewählten Bibliothek. Bei der Benennung eines Bibliothekselements müssen die folgenden Regeln beachtet werden:

- Der Name muß sich auf **8** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Weitere Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen (_) sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Jedem Bibliothekselement wird ein Kommentar zugeordnet, der bei der Erstellung des Elements eingegeben wird. Wenn das neue Element erstellt ist, müssen die folgenden Informationen eingegeben werden:

- seine Definition (für E/A-Konfigurationen),

- seine Parameter (für E/A-Karten),
- seine Anwender-Schnittstelle (für Funktionen und Funktionsbausteine).

Bei der Erstellung von "C"-Umrechnungen, "C"-Funktionen oder "C"-Funktionsbausteinen wird das komplette Modell ihres Quellcodes automatisch generiert.

Bestehende Elemente bearbeiten

Der Befehl "**Datei / Umbenennen**" wird benutzt, um den Namen oder Kommentar eines in der Liste ausgewählten Elements zu ändern. Der Befehl "**Datei / Kopieren**" wird benutzt, um das in der aktiven Bibliothek markierte Element in einem anderen Element derselben Bibliothek zu kopieren. Wenn das Zielelement bereits existiert, wird sein gesamter Inhalt überschrieben. Wenn das Zielelement nicht existiert, wird es automatisch erstellt. Der Befehl "**Datei / Löschen**" entfernt das ausgewählte Element aus der aktiven Bibliothek. Die folgenden Elementkomponenten werden von den Befehlen "**Umbenennen**", "**Kopieren**" und "**Löschen**" mitverarbeitet:

- technisches Datenblatt,
- komplette Definition (für E/A-Konfigurationen),
- Parameter (für E/A-Karten und -Baugruppen),
- Schnittstellendefinition (für Funktionen und Funktionsbausteine),
- Quellcode (für Funktionen und Funktionsbausteine, die in den IEC-Sprachen geschrieben wurden),
- Quellcode (für C-Umrechnungen, -Funktionen und -Funktionsbausteine).

⚠ Wenn das Element eine "C"-Umrechnung, eine "C"-Funktion oder ein "C"-Funktionsbaustein ist, wird sein Name durch die Befehle "**Umbenennen**" und "**Kopieren**" nicht automatisch im zugeordneten Quellcode aktualisiert.

⚠ Wenn das Element eine in einer IEC-Sprache geschriebene Funktion ist, wird der Name seines Returnparameters durch die Befehle "**Umbenennen**" und "**Kopieren**" nicht automatisch geändert.

Datenschutz durch Paßwörter

Der Befehl "**Datei / Paßwort**" wird benutzt, um ein Paßwort zum Schutz des in der offenen Bibliothek ausgewählten Elements zu bestimmen. Siehe Abschnitt "**Datenschutz durch Paßwörter**" am Ende des ersten Teils dieses Handbuchs für weitere Informationen über Paßwortebenen und Datenschutz. Paßwörter beziehen sich auf das ausgewählte Element und nehmen keinen Einfluß auf die anderen Elemente der ISaGRAF Bibliotheken.

Funktionen und Funktionsbausteine kompilieren

Wenn die Bibliothek der in den IEC-Sprachen geschriebenen Funktionen und Funktionsbausteine offen ist, wird der Befehl "**Prüfen (kompilieren)**" im "**Datei**"-Menü benutzt, um die Synatx des ausgewählten Elements zu prüfen und seinen Objektcode zu erstellen. Die in den IEC-Sprachen geschriebenen Funktionen und Funktionsbausteine müssen fehlerfrei kompiliert werden, bevor sie in den ISaGRAF Projekten benutzt werden können. Dieser Befehl ist wirkungslos, wenn eine andere Bibliothek ausgewählt ist.

Technische Datenblätter

Der Befehl "**Bearbeiten / Datenblatt**" wird benutzt, um das technische Datenblatt des in der aktiven Bibliothek ausgewählten Elements einzugeben. Die Eingabe erfolgt mit dem ISaGRAF Texteditor. Das technische Datenblatt ist die **Gebrauchsanleitung** eines Elements und wird vom Benutzer bei der Integration des Elements in ein ISaGRAF Projekt benutzt. Das technische Datenblatt sollte die hauptsächlichen Funktionen, eine genaue Beschreibung der Programmierschnittstelle und der Parameter, sowie den Kontext und die Begrenzungen des betreffenden Elements beinhalten.
Der Befehl "**Werkzeuge / Format des Datenblatts**" wird benutzt, um ein Standardtextformat für die Elemente der offenen Bibliothek zu definieren. Wenn das Datenblatt eines neuen Elements bearbeitet wird, wird dieses Format als Eingabemodell benutzt. Auf diese Weise kann die Bearbeitung der technischen Datenblätter optimiert werden.

Parameter

Die Parameter eines Elements beschreiben die **Schnittstelle** zwischen den Computeroperationen, die dieses Element ausführt, und der Benutzung des Elements in einer ISaGRAF Anwendung. Bei Bibliothekselementen unterschiedlichen Typs haben die Parameter unterschiedliche Bedeutungen.
Die Parameter einer E/A-Konfiguration definieren einen kompletten Satz an E/A-Karten und die Vorgabenamen der Variablen für die E/A-Kanäle. Die Parameter einer E/A-Karte oder einer komplexen E/A-Baugruppe definieren die physikalische und logische Konfiguration der Karte. Die Parameter einer Funktion oder eines Funktionsbausteins definieren die Schnittstelle des Elements gemäß der ST-Funktionsaufrufkonventionen. Umrechnungsfunktionen haben keine Parameter, weil sie festgelegte Standard-Schnittstellen benutzen.

C-Quellcode

Die ISaGRAF Workstation gibt dem Programmierer die Möglichkeit, den Quellcode der Umrechnungen, Funktionen und Funktionsbausteine der Bibliothek zu verwalten. Der Quellcode der Funktionen und Funktionsbausteine, die in den IEC-Sprachen geschriebenen wurden, besteht aus einem Text oder Diagramm, das in einer dieser Funktion zugeordneten Sprache geschrieben wurde. Der Quellcode von "C"-Komponenten ("C"-Funktionen, "C"-Funktionsbausteinen und Umrechnungsfunktionen) besteht aus zwei getrennten Dateien: Der **Quellcode-Vorsatz** enthält die genaue Definition der **Schnittstelle**, entsprechend der Parameterdefinition des Elements. In der **Quellcode**-Datei wird die operationelle Implementierung des Elements beschrieben.
Die ISaGRAF Workstation generiert die Quellcode-Datei, wenn ein neues Bibliothekselement erstellt wird. Außerdem aktualisiert sie den Quellcode-Vorsatz gemäß der Parameterdefinition. Der Programmierer kann den ISaGRAF Texteditor benutzen, um die Quellcode-Datei zu vervollständigen.

Bibliothekselemente archivieren

Der Befehl "**Werkzeuge / Archivieren**" startet den ISaGRAF Archivmanager, um Bibliothekselemente zu sichern oder wiederherzustellen. Bevor man diesen Befehl benutzen kann, muß eine Bibliothek ausgewählt werden. Im Archivmanager erscheint jeweils nur die Elementliste einer Bibliothek.

A.22.2. E/A-Konfiguration

Mit Hilfe der ISaGRAF Bibliothek der E/A-Konfigurationen können neue ISaGRAF Projekte problemlos mit bereits definierten E/A-Konfigurationen initialisiert werden. Eine E/A-Konfiguration definiert:

- einen Satz an E/A-Karten
- Vorgabewerte für die Parameter der E/A-Karten
- Vorgabenamen für die E/A-Kanäle

Wenn ein neues ISaGRAF Projekt mit einer E/A-Konfiguration der Bibliothek erstellt wird, erfolgt der entsprechende E/A-Anschluß automatisch und die E/A-Variablen, die den Kanalnamen entsprechen, werden automatisch im Datenverzeichnis des Projekts deklariert.

Die Definition einer E/A-Konfiguration erfolgt mit dem ISaGRAF E/A-Verdrahtungswerkzeug (das auch in den Projekten benutzt wird). Siehe Abschnitt "E/A-Verdrahtung" in diesem Handbuch für eine ausführliche Beschreibung dieses Werkzeugs. Wenn eine neue Karte in eine Konfiguration eingefügt wird, werden alle Kanäle der neuen Karte mit Standard-Vorgabenamen deklariert. Der Standard-Vorgabename eines E/A-Kanals hat das folgende Format:

<richtung><typ><steckplatznummer>_<kanalnummer>

Das erste Zeichen kennzeichnet die Richtung des E/A-Kanals:
"**I**"..... Eingangskanal
"**Q**" ..... Ausgangskanal

Das zweite Zeichen kennzeichnet den Typ des E/A-Kanals:
"**X**"..... Boolesch
"**D**"..... analog
"**M**" ..... Zeichenkette

Hier einige Beispiele eines Standard- E/A-Kanalnamens:
IX0_7..... Boolescher Eingang - Karte #0 - Kanal #7
QD2_4..... ganzzahliger Ausgang - Karte #2 - Kanal #4

Der Befehl "E/A-Kanäle anschließen" des E/A-Verdrahtungseditors wird benutzt, um den vorgegebenen Namen, der einem E/A-Kanal zugeordnet ist, zu modifizieren.

A.22.3. Komplexe E/A-Baugruppen

Alle Kanäle einer einzelnen Karte haben den gleichen Typ (Boolesch, analog oder Zeichenkette) und die gleiche Richtung (Eingang oder Ausgang). Eine komplexe E/A-Baugruppe ist eine E/A-Einrichtung mit Kanälen unterschiedlichen Typs oder unterschiedlicher Richtungen. Eine komplexe E/A-Baugruppe wird als eine Liste einzelner E/A-Karten dargestellt. Sie belegt nur einen Steckplatz im Baugruppenträger der E/A-Verdrahtung.

Um eine komplexe E/A-Baugruppe zu definieren, muß der Benutzer die Liste der einzelnen Karten definieren, aus denen sich die E/A-Baugruppe zusammensetzt.

Außerdem muß er die Parameter jeder einzelnen Karte eingeben. Die Liste der einzelnen E/A-Karten wird in einem Dialogfeld eingegeben.

Mit Hilfe der Schaltfläche "**Hinzufügen**" kann der Benutzer eine einzelne Karte am Ende der aktuellen Liste hinzufügen. Die Schaltfläche "**Einfügen**" wird benutzt, um eine neue, einzelne Karte vor der in der Liste ausgewählten Karte einzufügen. Die Schaltfläche "**Löschen**" nimmt eine einzelne, in der Liste ausgewählte Karte aus der Liste. Die Schaltflächen "**Umbenennen**" und "**Parameter**" werden benutzt, um den Namen und die Parameter der einzelnen, ausgewählten Karte zu ändern. Siehe folgenden Abschnitt für eine ausführliche Erläuterung der Parameter einzelner Karten. Eine komplexe E/A-Baugruppe kann bis zu **16** einzelne E/A-Karten enthalten. Der Name einer einzelnen Karte (innerhalb einer E/A-Baugruppe) darf nicht länger als **8** Zeichen sein.

A.22.4. E/A-Karten

Die ISaGRAF Bibliothek der E/A-Karten definiert eine Standard-Schnittstelle zwischen den Variablen der Anwendung und der Ziel-Hardware. In der Programmierungsphase einer Anwendung werden alle E/A-Variablen mit den Kanälen der E/A-Zielkarten verdrahtet. Eine ISaGRAF E/A-Karte wird durch einen **Namen** und einen "**OEM-Schlüsselcode**", der ihren **Hersteller** kennzeichnet, definiert. Andere E/A-Kartenparameter beschreiben die Topologie der E/A-Karte (Anzahl der Kanäle, Richtung und Typ der Kanäle) und ihre Hardware- und Softwarekonfiguration.

E/A-Kartenparameter

Bei den E/A-Karten unterscheidet man zwischen zwei Parametertypen: die allgemeinen Parameter, die für alle Karten der ISaGRAF Bibliothek definiert werden, und die vom Kartenhersteller gelieferten OEM-Parameter, die spezifisch für die Implementierung einer Karte sind. Die allgemeinen Parameter werden im oberen Teil des Dialogfelds für die Eingabe der E/A-Kartenparameter eingegeben. Diese Parameter (und der Name der E/A-Karte) kennzeichnen die Standard-Schnittstelle einer ISaGRAF E/A-Karte.
Der "**OEM-Schlüsselcode**" ist eine einfache Zahl, die den **Hardware-Hersteller** kennzeichnet. Alle Karten eines bestimmten Herstellers besitzen den gleichen OEM-Schlüsselcode. Der OEM-Schlüsselcode ist eine **vorzeichenlose Zahl auf 16 Bits** und wird im **Hexadezimal**format eingegeben. Der OEM-Schlüsselcode von **CJ International** ist "**1**".
Die hauptsächlichen Parameter definieren die Topologie der E/A-Karte. Die **Anzahl der Kanäle** definiert die Anzahl der verfügbaren Kanäle einer Karte. Der **Typ** der Karte ist der Typ der Variablen, die auf den Kanälen einer Karte verdrahtet werden können. Die **Richtung** definiert, ob die an die Karte angeschlossenen Variablen **Eingang**s- oder **Ausgang**svariablen sind.

Hinweis: E/A-Variablen unterschiedlichen Typs oder verschiedener Richtungen können nicht auf einer einzelnen ISaGRAF E/A-Karte verdrahtet werden. Hierzu wird eine komplexe E/A-Baugruppe benötigt.

OEM-Parameter

Die OEM-Parameter werden im unteren Teil des Dialogfelds für die Definition der E/A-Kartenparameter eingegeben. Diese Parameter werden vom Hersteller der E/A-Karte definiert und sind karten-spezifisch. Eine E/A-Karte kann bis zu **16** OEM-Parameter besitzen. Es gibt auch Karten ohne OEM-Parameter. Mit dem ISaGRAF Bibliotheksmanager kann der Hardware-Hersteller die Kennzeichnung und das Format eines jeden dieser Parameter definieren, sowie die Art, wie diese Parameter vom Programmierer eingegeben werden müssen.
Das linke Feld enthält die Liste der OEM-Parameter. Jeder Parameter ist mit einem **Namen** und einer logischen **Nummer** zwischen **0** und **15** gekennzeichnet. Rechts erscheint eine ausführliche Beschreibung des in der Liste ausgewählten Parameters. Wählen Sie einen Parameter in der Liste, um auf seine Beschreibung zuzugreifen. Klicken auf die Schaltfläche "**Löschen**" löscht die Parameterbeschreibung und nimmt den Parameter aus der Liste. Achtung: Dieser Befehl kann nicht "widerrufen" werden.
Der Parametername wird benutzt, um das entsprechende Eingabefeld bei der E/A-Verdrahtung einer Karte zu identifizieren, wenn das Feld von einem Automatisierungstechniker definiert werden muß. Bei der Parameterbenennung sollten die folgenden Regeln beachtet werden:

- Der Name muß sich auf **16** Zeichen begerenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Weitere Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen '_' sein.

Der Typ eines Parameters definiert das **interne Format** des Parameters, sowie sein **Eingabeformat** bei der E/A-Verdahtung. Die folgenden internen Formate sind verfügbar:

word ..... vorzeichenloses Wort auf 16 Bits
long ..... vorzeichenloses Wort auf 32 Bits
word hexa ..... vorzeichenloses Wort auf 16 Bits
long hexa ..... vorzeichenloses Wort auf 32 Bits
boolean ..... vorzeichenloses Wort auf 16 Bits (nur unterstes Bit wird benutzt)
character ..... vorzeichenloses Wort auf 16 Bits (nur unterstes Byte wird benutzt)
string ..... Feld von 16 Bytes, das eine in null endende Zeichenkette einschließt
float ..... Gleitpunktwert auf 32 Bits mit einfacher Präzision

Folgende Eingabeformate sind verfügbar:

word ..... vorzeichenloses Dezimalwort
long ..... langes Dezimalwort
word hexa ..... vorzeichenloses Hexadezimalwort
long hexa ..... vorzeichenloses langes Hexadezimalwort
boolean ..... "true" oder "false"
character ..... einzelnes Zeichen
string ..... Ascii-Zeichenkette (max. 15 Zeichen)
float ..... Gleitpunktwert mit einfacher Präzision

Das Feld "**Zugriff**" wird benutzt, um zu definieren, ob der Endanwender auf den Parameter zugreifen kann. Wenn die Option "**Veränderbar**" aktiviert wird, erscheint der Parameter bei der E/A-Verdrahtung als ein Eingabefeld. Der OEM-Parameter-Vorgabewert wird als Vorgabe bei der Bearbeitung eines Parameters benutzt. Wenn die Option "**Versteckt**" aktiviert wird, ist der Parameter konstant und wird nicht im Dialogfeld der E/A-Verdahtung angezeigt. Der OEM-Parameter-Vorgabewert definiert den Wert des konstanten Parameters. Die Option "**Nur lesen**"

kennzeichnet, daß der Parameter für den Endanwender sichtbar ist, jedoch nicht modifiziert werden kann. Sein Vorgabewert wird als konstanter Wert benutzt.

A.22.5. In anderen IEC-Sprachen geschriebene Funktionen und F-Bausteine

ISaGRAF verwaltet eine Bibliothek von Funktionen und Funktionsbausteinen, die in den IEC-Sprachen geschrieben wurden. Die verfügbaren Sprachen für die Beschreibung dieser Funktionen oder Funktionsbausteine sind **FBS** (Funktionsbaustein-Sprache), **KOP** (Kontaktplan), **ST** (Strukturierter Text) und **AWL** (Anweisungsliste). Die KOP- und FBS-Sprachen können in einem Diagramm gemischt vorkommen. Die **AS-**Sprache (Ablaufsprache) kann nicht für die Beschreibung von Funktionen und Funktionsbausteinen der Bibliothek benutzt werden. Die Sprache für die Beschreibung eines Bibliothekselements wird bei seiner Erstellung gewählt und kann später nicht mehr geändert werden.

Kompilierung

Die in der Bibliothek definierten Funktionen und Funktionsbausteine müssen kompiliert (geprüft) werden, bevor sie in einem ISaGRAF Projekt benutzt werden können. Ansonsten sind für die Funktionen und Funktionsbausteine der Bibliothek keine weiteren Änderungen nötig. Die Bibliothekselemente werden automatisch in einem Auswahlfeld angezeigt, wenn der grafische KOP/FBS-Editor innerhalb eines Projekts benutzt wird.

⚠ Eine in der Bibliothek definierte Funktion kann andere Funktionen der Bibliothek aufrufen. Das ISaGRAF System **unterstützt** jedoch **keine Rekursivität** bei Funktionsaufrufen. Ein in einer IEC-Sprache geschriebener Funktionsbaustein kann keine anderen Funktionsbausteine aufrufen (egal, ob die Bausteine in einer IEC-Sprache oder in der "C"-Sprache geschriebenen wurden).

Eingabe des Quellcodes

Der Quellcode einer Bibliotheksfunktion oder eines Bibliotheksbausteins wird mit den ISaGRAF Standardwerkzeugen eingegeben: mit dem Grafikeditor bei KOP- oder FBS-Programmen, mit dem Texteditor bei ST- oder AWL-Programmen. Siehe betreffende Abschnitte in diesem Handbuch für eine ausführliche Beschreibung dieser Werkzeuge. Der ISaGRAF Codegenerator kann direkt von den Fenstern der Grafik- und Textbearbeitung aufgerufen werden, um den Quellcode der Bibliotheksfunktionen und -funktionsbausteine zu kompilieren.

Datenverzeichnis der lokalen Variablen

Bibliotheksfunktionen und -funktionsbausteine können lokale Variablen und Definitionen besitzen. Um auf die Variablendeklaration zuzugreifen, benutzt man die Befehle "**Datenverzeichnis**" im "**Datei**"-Menü des Editorfensters, wenn der Quellcode der Funktion bearbeitet wird.

⚠ Bibliotheksfunktionen und -funktionsbausteine können nicht auf globale Variablen oder Funktionsbaustein-Instancen zugreifen. Die lokalen Variablen sollten im Rumpf der Funktion initialisiert werden.

Lokale Variablen eines Funktionsbausteins, der in einer IEC-Sprache geschrieben wurde, werden jedesmal kopiert (instanziiert), wenn der Baustein in einem Projekt benutzt wird. Lokale Variablen einer Instance behalten ihre Werte von einem Aufruf zum anderen bei.

Definition der Schnittstelle

Funktionen und Funktionsbausteine können bis zu **32** Parameter besitzen (Ein- und Ausgänge). Eine Funktion hat nur einen einzigen Returnparameter, der den gleichen Namen wie die Funktion tragen muß, gemäß der Schreibkonventionen der ST-Sprache.
Links oben erscheint die Liste der Parameter in der Reihenfolge des Aufrufmodells: zuerst die Aufrufparameter, zuletzt die Returnparameter. Im unteren Teil des Fensters erscheint die ausführliche Beschreibung des in der Liste ausgewählten Parameters. Für Parameter kommen alle ISaGRAF Datentypen in Frage. Die Returnparameter müssen nach den Aufrufparametern in der Liste erscheinen. Bei der Parameterbenennung müssen die folgenden Regeln beachtet werden:

- Der Name muß sich auf **16** Zeichen begrenzen.
- Das erste Zeichen muß ein **Buchstabe** sein.
- Weitere Zeichen können **Buchstaben**, **Zahlen** oder Unterstrichzeichen '_' sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Der Befehl "**Einfügen**" wird benutzt, um einen neuen Parameter vor dem ausgewählten Parameter einzufügen. Der Befehl "**Löschen**" wird benutzt, um den ausgewählten Parameter zu löschen. Der Befehl "**Sortieren**" sortiert die Parameter automatisch, so daß die Returnparameter am Ende der Liste erscheinen.

A.22.6. "C"-Funktionen und -Funktionsbausteine

"C"-Funktionen und -Funktionsbausteine sind **Rechenfunktionen**, die von der Automatisierungsanwendung gemäß der Funktionsaufruf-Schnittstelle der ST-Sprache aufgerufen werden.
Funktionen sind **synchrone** Verarbeitungen. Während der Ausführung einer Funktion wird die ISaGRAF Zielanwendung unterbrochen. Funktionsbausteine ordnen Operationen versteckte statische Daten zu. (Der Funktionsbaustein "counter" stellt eine Zähloperation, sowie das Zählergebnis dar.) Funktionen und Funktionsbausteine können benutzt werden, um die Kapazitäten der Standard-Automatisierungssprachen zu erweitern oder um auf Systemresourcen zuzugreifen.

Das Dialogfeld für die Parameterdefinition wird benutzt, um die Namen und Typen sämtlicher Aufruf- und Returnparameter einer Funktion oder eines Funktionsbausteins zu definieren. Die Befehle im Menü "**Bearbeiten**" werden benutzt, um die Parameter der ausgewählten Funktion oder des ausgewählten Bausteins zu definieren. Eine Funktion kann bis zu **31** Aufrufparameter besitzen und hat stets **einen einzigen** Returnparameter. Ein Funktionsbaustein kann bis zu **32** Parameter besitzen, beliebig zusammmengesetzt aus Aufruf- und Returnparametern. Die Entsprechung zwischen ISaGRAF Typen und "C"-Typen ist wie folgt:

BOOLESCH	unsigned long	vorzeichenloses Wort auf 32 Bits 1=true / 0=false

ANALOG	long	vorzeichenbehaftetes ganzzahliges Wort auf 32 Bits
REAL	float	Gleitpunktwert mit einfacher Präzision
TIMER	unsigned long	vorzeichenloses ganzzahliges Wort auf 32 Bits (Einheit = 1 ms)
ZEICHENKETTE	char *	Zeichenkette

Wenn Zeichenkettenwerte an "C"-Funktionen oder -Funktionsbausteine gegeben werden, dürfen sie keine Nullzeichen enthalten, weil Zeichenketten, die an einen "C"-Code gegeben werden, stets mit Null enden.
Siehe ISaGRAF Zielsystem-Benutzerhandbuch für weitere Informationen zur Verwaltung des "C"-Quellcodes einer Funktion oder eines Funktionsbausteins, sowie zur Integration eines neuen Elements in das ISaGRAF Zielsystem.

A.22.7. Umrechnungsfunktionen

Eine Umrechnungsfunktion ist eine "C"-Funktion, die vom ISaGRAF E/A-Manager jedesmal dann aufgerufen wird, wenn die analogen Variablen, die diese Umrechnung benutzen, vom Projekt erworben oder aktualisiert werden.
Die Funktion liefert die Beziehung zwischen dem **elektrischen** (auf einem Sensor gelesenen oder zu einem Aktuator gesendeten) Wert und dem **physikalischen** (in den Ausdrücken der Anwendung benutzten) Wert einer Variablen. Deshalb bestehen Umrechnungen aus zwei Teilen: Eingangsumrechnungen und Ausgangsumrechnungen. Der "C"-Quellcode dieser Umrechnungsfunktionen kann mit dem ISaGRAF Bibliotheksmanager verwaltet werden.
Umrechnungen können für **ganzzahlige** oder **reale** analoge Variablen benutzt werden. Aus diesem Grunde wird die Schnittstelle der Umrechnungsfunktionen mit Gleitpunktwerten definiert. Sie ist für alle Umrechnungsfunktionen die gleiche. Die "C"-Definition dieser Schnittstelle erfolgt in der Definitionsdatei "**TACN0DEF.H**".
Siehe ISaGRAF Zielsystem-Benutzerhandbuch für weitere Informationen zur Verwaltung des "C"-Quellcodes der Umrechnungsfunktionen, sowie zur Integration eines neuen Elements in das ISaGRAF Zielsystem.

A.23. Archivierungs-Utility benutzen

Die ISaGRAF Archivierungs-Utility ermöglicht die Sicherung der ISaGRAF Projekte und -Bibliotheken auf Disketten oder in einem Sicherungsverzeichnis. Der ISaGRAF Archivmanager ist ein Dialogfeld, das vom Fenster des ISaGRAF Projektmanagers oder Bibliotheksmanagers eingeleitet wird.

⚠ Um zuverlässige Archive zu erstellen, sollten die folgenden Richtlinien befolgt werden:

- Der Name und die Beschreibung des gesicherten Objekts sollte auf das Etikett der Diskette geschrieben werden.
- Projekte und Bibliotheken sollten nicht zusammen auf einer Diskette gesichert werden.
- Verschiedene Projekte sollten nicht zusammen auf einer Diskette gesichert werden.

A.23.1. Archivmanager aufrufen

Das Dialogfeld "Archivierung" wird mit dem Menübefehl "**Werkzeuge / Archivieren**" des Projektmanager-Fensters geöffnet. Hier können Projekte oder gemeinsame Daten gesichert und wiederhergestellt werden.
Das Dialogfeld "Archivierung" kann auch mit dem Befehl "**Werkzeuge / Archivieren**" des ISaGRAF Bibliotheksmanagers aufgerufen werden, wenn Elemente der im Fenster ausgewählten Bibliothek gesichert oder wiederhergestellt werden sollen.

Projekte

Ein Projekt wird stets zusammen mit allen Projektkomponenten (Quelldateien der Programme, Objektcode und ausführbarer Anwendungscode) in einer Archivdatei gesichert. Durch Aktivieren der Option "**Kompression**" kann der Umfang der Projektdatei reduziert werden.

Bibliothekselemente

Die Elemente der ISaGRAF Bibliotheken können individuell gesichert werden. Sämtliche Komponenten eines Bibliothekselements (technisches Datenblatt, Definition, Schnittstelle, Quellcode...) werden zusammen mit dem Element in einer Archivdatei gesichert.

Allgemeine Daten

Der Befehl "**Werkzeuge / Archivieren / Gemeinsame Daten**" im Fenster des Projektmanagers dient der Sicherung / Wiederherstellung der gemeinsamen Daten der ISaGRAF Workstation (Daten mit "allgemeinem Geltungsbereich"). Dieser Befehl hat keine Auswirkung auf die ISaGRAF Bibliotheken. Die folgenden Dateien können mit diesem Befehl kopiert werden:
common.eqv ..... allgemeine Definitionen (äquivalente Wörter)
oem.bat ..... anwenderdefinierte MS-DOS-Befehlsdatei.

Die Dateien werden einzeln in ihrem Originalformat auf der Archivdiskette gesichert. Diese Achivdateien werden nicht komprimiert.

A.23.2. Optionen

Der Pfad des ISaGRAF Archivs erscheint unten im Dialogfeld. Klicken Sie auf die Schaltfläche "Durchlaufen", um ein anderes Laufwerk und Archivverzeichnis aus den verfügbaren Laufwerken auszuwählen.

Wenn die Option "**Kompression**" aktiviert wird, werden alle während der "**Sicherung**" erstellten Archivdateien komprimiert. Diese Option ist nützlich, wenn umfangreiche Projektarchivdateien auf einer einzigen Diskette gespeichert werden sollen. Für die Archivierung der Bibliothekselemente ist die Kompression im allgemeinen nicht nötig. Bei der Wiederherstellung erkennt der ISaGRAF Archivmanager den Status einer Archivdatei (komprimiert oder nicht) automatisch. Die Option "**Kompression**" nimmt keinen Einfluß auf die "**Wiederherstellung**".

A.23.3. Sichern und Wiederherstellen

In der "**Workstation**"-Liste erscheinen die aktuellen Objekte der auf der Festplatte installierten ISaGRAF Workstation (links). Die "**Archiv**"-Liste zeigt die gesicherten Objekte des spezifizierten Archivverzeichnisses an (rechts).

- ***Sichern***

  Um ein Objekt in einem Archiv zu sichern, wählen Sie es in der **linken** Liste aus (Objekte der ISaGRAF Workstation) und klicken auf die Schaltfläche "**Sichern**". Sie können mehrere Objekte in der Liste auswählen. Die Schaltfläche "**Sichern**" ist nicht aktiv, wenn ein Element in der **rechten** Liste ausgewählt ist (Wiederherstellungsmodus).

- ***Wiederherstellen***

  Um ein Objekt aus einem Archiv in die ISaGRAF Workstation zu kopieren, wählen Sie das Objekt in der **rechten** Liste (Archivobjekte) und klicken auf die Schaltfläche "**Wiederherstellen**". Sie können mehrere Objekte in der Liste auswählen. Die Schaltfläche "**Wiederherstellen**" ist nicht aktiv, wenn ein Element in der **linken** Liste ausgewählt ist (Sicherungsmodus).

A.23.4. Archivdateien

Der ISaGRAF Archivmanager erstellt eine einmalige Archivdatei für jedes gesicherte Objekt, die den gleichen Namen wie das Objekt trägt. Der Typ der Datei ist aus dem Suffix erkenntlich. Folgende Suffixe werden benutzt:

.pia ..... Projekt
.bia ..... E/A-Karte
.iia ..... Funktion in IEC-Sprache
.aia ..... Funktionsbaustein in IEC-Sprache
.uia ..... C-Funktion
.fia ..... C-Funktionsbaustein
.cia ..... C-Umrechnungsfunktion
.ria ..... E/A-Konfiguration
.xia ..... E/A-Baugruppe

A.24. Eine komplette Dokumentation ausdrucken

Der ISaGRAF Dokumentations-Generator dient dem Aufbau und Ausdruck einer kompletten Dokumentation für das ausgewählte Projekt. Der Dokumentations-Generators wird mit dem Befehl "**Drucken**" im Menü "**Projekt**" des Projektmanagers oder der Programmfenster aktiviert. Der Dokumentations-Generator kann auch mit dem Befehl "**Drucken**" der ISaGRAF Editoren gestartet werden, um den Inhalt eines einzelnen ISaGRAF Dokuments auszudrucken. In beiden Fällen sind folgende Funktionalitäten verfügbar:

Die Befehle im Menü "**Bearbeiten**" werden für die Definition der Projektelemente benutzt, die in die Dokumentation eingefügt werden sollen. Sämtliche Projektdaten (Programme, Variablen, Optionen, E/A-Verdrahtung...) können in die Projektdokumentation aufgenommen werden. Daten, die andere Projekte oder die ISaGRAF Bibliotheken betreffen, erscheinen nicht in der Projektdokumentation.

Der Befehl "**Datei / Drucken**" generiert die Dokumentation gemäß des spezifizierten Inhaltsverzeichnisses und sendet sie zum Drucker. Der Druckbefehl kann mehrere Minuten für den Aufbau und die Formatierung der Dokumentation in Anspruch nehmen. Es wird empfohlen, abzuwarten, bis der Druckbefehl im Fenster des ISaGRAF Dokumentations-Generators beendet ist, bevor andere Befehle der ISaGRAF Workstation eingeleitet werden. Der Aufbau einer kompletten Dokumentation kann sehr viel Speicherplatz auf der Festplatte benötigen. Wenn nicht genug Speicherplatz vorhanden ist, wird eine Fehlermeldung angezeigt. In diesem Fall muß Speicherplatz auf der Festplatte geschaffen werden, indem Dateien gelöscht werden oder der Umfang der Dokumentation reduziert wird. Wenn der Befehl "**Drucken**" eingeleitet wird, erscheint ein Dialogfeld, in dem eine Beschreibung des aktuellen Druckbefehls eingegeben werden kann. Diese Angaben werden in einer Protokolldatei gespeichert und auf der ersten Seite der aktuellen Dokumentation (sowie zukünftiger Dokumenationen) gedruckt

A.24.1. Inhaltsverzeichnis persönlich gestalten

Mit den Befehlen im Menü "**Bearbeiten**" kann das "Inhaltsverzeichnis" einer Dokumentation definiert werden. Verschiedene Befehle können benutzt werden, um das vorgegebene Inhaltsverzeichnis (mit allen Projektkomponenten) zu benutzen, ein spezielles Inhaltsverzeichnis (mit ausgesuchten Komponenten) zu erstellen oder Elemente im Inhaltsverzeichnis zu verschieben (das Inhaltsverzeichnis zu verändern).

Vorgabeliste

Der Befehl "**Standardliste**" im Menü "**Bearbeiten**" definiert das vorgegebene Inhaltsverzeichnis mit allen Projektkomponenten für die Dokumentation. Die Standardliste beinhaltet:

- Projektkopf
- Hierarchische Baumstruktur (Programmverknüpfungen)
- Quellcode eines beliebigen Programms
- Protokolldatei eines beliebigen Programms

- Allgemeine Definitionen
- Globale Definitionen
- Lokale Definitionen eines beliebigen Programms
- Globale Variablen
- Lokale Variablen eines beliebigen Programms
- Anwendungsoptionen
- E/A-Verdrahtung
- Variablenlisten
- Umrechnungstabellen
- Verkürzte Crossreferenzen
- Detaillierte Crossreferenzen
- Zusammenfassung der Deklarationen
- Netzwerkadresskarte
- Projektprotokoll

Das Inhaltsverzeichnis kann mit dem Befehl "**Datei / Sichern**" auf der Festplatte gesichert werden. Dieser Befehl ist nicht verfügbar, wenn der Dokumentations-Generator von einem ISaGRAF Editor aus eingeleitet wird, um ein einzelnes Dokument auszudrucken.

Ausschneiden und Einfügen

Benutzen Sie die Befehle "**Bearbeiten / Ausschneiden**" und "**Bearbeiten / Einfügen**", um Elemente in der Liste zu verschieben und die Anordnung der Elemente zu verändern. Der Dokumentations-Generator ermöglicht die Auswahl mehrerer Elemente im Inhaltsverzeichnis, so daß eine Gruppe von Elementen ausgeschnitten und eingefügt werden kann.

Inhaltsverzeichnis löschen

Benutzen Sie den Befehl "**Bearbeiten / Löschen**", um das Inhaltsverzeichnis zu löschen, so daß es durch Einfügen einzelner Elemente ganz neu aufgebaut werden kann.

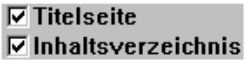
Elemente in das Inhaltsverzeichnis einfügen

Wenn der Befehl "**Bearbeiten / Einfügen**" eingeleitet wird, erscheint eine Dialogfeld. Hier kann der Benutzer Elemente (Projektkomponenten) in das Inhaltsverzeichnis einfügen.

Im Listenfeld "**Programm**" kann ein Programmname für programm-lokale Elemente ausgewählt werden. Klicken Sie auf die Schaltfläche "**Hinzufügen**" um das ausgewählte Element in das Inhaltsverzeichnis einzufügen. Ein Element kann jeweils nur einmal im Inhaltsverzeichnis vorkommen.

A.24.2. Optionen

Mit den Befehlen im Menü "**Optionen**" kann das Format der generierten Dokumentation definiert und persönlich gestaltet werden.
Weitere Optionen sind unmittelbar im Fenster des Dokumentations-Generators als Schaltflächen verfügbar:



Wenn die Option "**Titelseite**" aktiviert wird, wird eine Titelseite mit dem Titel des Projekts und dem Druckprotokoll am Anfang eines Dokuments gedruckt. Wenn diese Option nicht aktiviert wird, beginnt der Ausdruck mit der ersten Seite.
Wenn die Option "**Inhaltsverzeichnis**" aktiviert wird, wird ein Inhaltsverzeichnis am Ende des Dokuments gedruckt.
Beide Optionen sind deaktiviert, wenn der Dokumentations-Generator mit dem Befehl "**Drucken**" eines ISaGRAF Editoren (Programm, Datenverzeichnis...) gestartet wird.

AS-Netzwerke

Mit der Option "**AS-Ebenen trennen**" kann für jedes AS-Programm zuerst die Ebene 1 (Netzwerk und Kommentare) und dann die Programmierung der Ebene 2 gedruckt werden. Wenn diese Option deaktiviert wird, erscheinen die Ebenen 1 und 2 zusammen im ausgedruckten Dokument.

Seitenformat

Benutzen Sie den Befehl "**Seitenumbruch**" im Menü "**Optionen**", um die hauptsächlichen Parameter für den Seitenumbruch der Dokumentation zu definieren. Folgende Parameter können bestimmt werden:

- **Linker Rand**: (1 oder 2 cm, oder kein Rand)
- **Rahmen**: Wenn diese Option aktiviert wird, werden alle Seiten mit einem Rahmen gedruckt.

Titelseite (Modell)

Mit dem Befehl "**Fußzeile**" im Menü "**Optionen**" kann der Inhalt der Fußzeile, die am Ende einer jeden Seite gedruckt wird, definiert werden. Das Standardformat der Fußzeile sieht so aus:

C/A	Text1	ISaGRAF - Projekt 'PrName'	Datum
	Text2		
	Text3	**Benutzerdefinierter Titel**	Seite

Die erste Titelzeile (mit dem Namen des ISaGRAF Projekts), das aktuelle Datum und die Seitenzahl werden automatisch vom Dokumentations-Manager generiert und können nicht modifiziert werden.
Die drei Textzeilen links (Text1, Text2 und Text3) und die zweite Titelzeile können frei vom Benutzer definiert werden. Das Logo im linken Feld kann ebenfalls geändert werden. Wenn Sie ein anderes Logo benutzten möchten, müssen Sie den Pfadnamen einer Bitmap-Bilddatei (**.BMP**) eingeben. Das Bild kann beliebig groß sein und wird beim Ausdruck je nach Seitengröße und Druckauflösung reduziert oder vergrößert. Klicken auf das Logofeld im Dialogfeld zeigt das neue Logo an. Die Bilddatei muß sich auf der Festplatte befinden (im angegebenen Verzeichnis und unter dem spezifizierten Pfadnamen), wenn der Befehl "**Drucken**" eingeleitet wird.

Auswahl der Schriftarten

Die Befehle "**Schriftart Text**" und "**Schriftart Titel**" im Menü "**Optionen**" werden benutzt, um die Schriftarten für die Titel und den Text der Dokumentation zu bestimmen. Die Größe und der Stil der Zeichen für Titel und Text können auch definiert werden. Die Auswahl der Schriftarten erfolgt im Windows-Standard-Dialogfeld. Sämtliche Texte (Textprogramme, Namen in den Diagrammen...) werden in der ausgewählten Größe, Stil und Schriftart gedruckt. Nur die Titel werden in der für die Titel ausgewählten Schriftart gedruckt.
Wenn keine Schriftart gewählt wird, wird die vorgegebene Schriftart des Druckers benutzt, und zwar in den folgenden Stilarten:

- "Normal" für Texte und Namen in den Diagrammen
- "Fett" für die Titel

A.25. Datenschutz durch Paßwörter

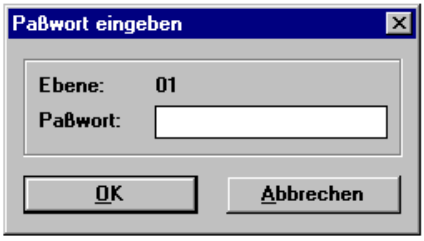
Die ISaGRAF Workstation besitzt ein komplettes Datenschutzsystem, mit dem der Benutzer Projekte und Bibliothekselemente durch Paßwörter schützen kann. Unter Bibliothekelementen versteht man E/A-Konfigurationen, E/A-Karten oder komplexe Baugruppen, in den IEC-Sprachen geschriebene Funktionen und Funktionsbausteine oder "C"-Funktionen, -Funktionsbausteine und -Umrechnungen. Paßwörter-Datenbänke sind Projekten oder Bibliothekselementen dediziert und können nicht von mehreren Objekten gemeinsam benutzt werden.

Schutzebenen

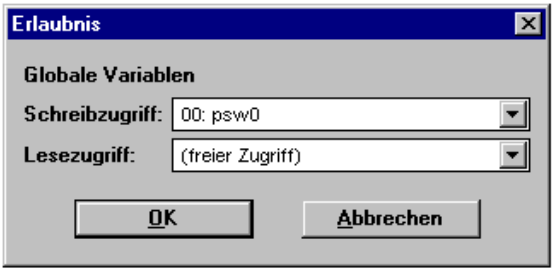
Innerhalb eines Projekts oder Bibliothekselements können bis zu **16** Zugriffsebenen definiert werden, die unterschiedlichen Paßwörtern entsprechen. Die Zugriffsebenen sind in einem Hierarchiesystem angeordnet und von **0** bis **15** nummeriert. Die höchste Zugriffsebene ist die Nummer **0**. Wenn der Benutzer ein Paßwort kennt, kann er auf alle Elemente, die durch diese Zugriffsebene geschützt werden, sowie auf alle durch niedrigere Zugriffsebenen geschützten Elemente, zugreifen. Jeder elementare Befehl oder Datensatz eines Projekts oder eines Bibliothekselements kann einzeln durch eine Zugriffsebene geschützt werden. Der Befehl "Anwendungscode entwickeln" der ISaGRAF Menüs kann beispielsweise getrennt geschützt werden. Unter elementaren Daten versteht man ein Programm, eine Optionsliste, das technische Datenblatt eines Bibliothekselements, etc...

Datenschutz definieren

Der Befehl "**Datenschutz**" in den ISaGRAF Menüs wird für die Definition von Paßwörtern und Zugriffsebenen für ein Projekt oder Bibliothekselement benutzt. Dieser Befehl wird in den Menüs des ISaGRAF Projektmanagers (für ein Projekt) oder vom ISaGRAF Bibliotheksmanager aus (für ein Bibliothekelement) aufgerufen. Wenn dieser Befehl zum ersten Mal eingeleitet wird, ist kein Paßwort erforderlich. Wenn schon Paßwörter definiert wurden, muß der Benutzer das höchste ihm bekannte Paßwort eingeben, bevor er auf diesen Befehl zugreifen kann. Paßwörter und geschützte Elemente der höheren Ebenen können nicht modifiziert werden. Mit dem Befehl "**Datenschutz**" kann der Benutzer Paßwörter für die verschiedenen Zugriffsebenen definieren und elementare Befehle oder Datensätze innerhalb der definierten Ebenen schützen.
Die Paßwörter für die Zugriffsebenen werden durch Doppelklicken auf eine Zeile in der oberen Liste eingegeben. Für die Eingabe wird das folgende Dialogfeld benutzt:



In der unteren Liste erscheinen die verschiedenen Elemente (Daten oder Funktionen), die geschützt werden können, sowie die aktuelle Zugriffsebene, die entweder einem "Lesezugriff" oder einem "unbeschränkten Zugriff" zugeordnet ist. Wenn man einer Schutzebene den "Lesezugriff" zuordnet, kann man vermeiden, daß unbefugte Benutzer Dokumente öffnen oder ausdrucken.
Doppelklicken Sie auf eine Zeile in der unteren Liste, um dem ausgewählten Element oder Data eine Erlaubnis zuzuordnen. Das folgende Dialogfeld wird angezeigt:



Beide Felder können auf "freien Zugriff" oder auf eine durch ein Paßwort definierte Datenschutzebene gesetzt werden. Es ist nicht möglich, einer Ebene eine unbeschränkte Zugriffserlaubnis zuzuordnen, wenn sie eine niedrigere Priorität hat als die im Feld "Lesezugriff" ausgewählte Ebene.
Hinweis: Bei bestimmten Dokumenten, die normalerweise immer in der ISaGRAF Workstation sichbar sind, wie dem Projektkopf, ist es nicht möglich, den Lesezugriff durch ein Paßwort zu schützen.

Zugriff auf geschützte Daten

Bei der Inbetriebnahme der Workstation ist kein Paßwort oder Benutzername erforderlich. Wenn der Benutzer versucht, auf geschützte Daten oder Funktionen zuzugreifen, wird er aufgefordert, das entsprechende Paßwort in einem Dialogfeld einzugeben.
Wenn der Benutzer das erforderliche Paßwort (oder das Paßwort einer höherer Zugriffsebene) eingibt, kann er normal mit der Bearbeitung fortfahren. Jedesmal, wenn der Benutzer ein Paßwort eingibt, wird es gespeichert, so daß später keine erneute Eingabe erforderlich ist. Gespeicherte Paßwörter werden bewahrt, wenn ein ISaGRAF Werkzeug von einem anderen ISaGRAF Werkzeug aus gestartet wird (z. B. wenn der Projektmanager den Programm-Manager startet). Gespeicherte Paßwörter gehen verloren, wenn das letzte ISaGRAF Fenster geschlossen wird. Paßwörter, die während der Projektbearbeitung, bei der Benutzung des Bibliotheksmanagers oder des Archivmanagers, eingegeben werden, können nicht gemeinsam benutzt werden. Wenn der Benutzer ein falsches Paßwort für eine ausgewählte Funktion eingibt, kann er diese Funktion nicht einleiten.

Verknüpfungen mit dem Archivmanager

Wenn ein Objekt (Projekt oder Bibliothekselement) archiviert wird, wird das Datenschutzelement "**Im Archiv sichern**" aufgerufen. Dies entspricht dem Datenschutzsystem, das dem Objekt in der Workstation (Festplatte) zugeordnet ist. Wenn das Objekt schon existiert, wird das Datenschutzsystem des Objekts im

Archiv nicht geprüft. Der Befehl "**Sichern**" des ISaGRAF Archivmanagers sichert die Datenschutzinformationen mit dem Objekt im Archiv.
Wenn ein Objekt, welches schon in der Workstation (Festplatte) existiert, wiederhergestellt wird, wird das Datenschutzelement "**Mit einem Archiv überschreiben**" aufgerufen. Dies entspricht dem Datenschutzsystem, das dem Objekt in der Workstation (Festplatte) zugeordnet ist. Das Datenschutzsystem des Objekts wird im Archiv nicht geprüft. Wenn dieser Befehl aktiviert wird, ersetzen die wiederhergestellten Datenschutzinfomationen die auf der Festplatte bestehenden.

Individueller Datenschutz für Variablen und E/A-Kanäle

Die ISaGRAF Workstation verfügt über ein komplettes Datenschutzsystem, das sich auf eine Paßwort-Hierarchie stützt. Die Variablendeklaration und die E/A-Verdrahtung können global durch Paßwörter geschützt werden. Zusätzlich ermöglicht ISaGRAF individuellen Schutz einzelner Variablen oder E/A-Kanäle. Dies setzt voraus:
- daß schon Paßwörter im Paßwortsystem definiert wurden (Befehl "**Projekt** / **Paßwort**" im Fenster des Projektmanagers), so daß Datenschutzebenen für den individuellen Datenschutz verfügbar sind.
- daß der individuelle Datenschutz auf einer höheren Prioritätsebene liegt als der globale Datenschutz.

Individuell geschützte Variablen oder E/A-Kanäle erscheinen im Fenster der Variablendeklaration oder der E/A-Verdrahtung mit einem kleinen Symbol neben ihrem Namen:
Benutzen Sie die Befehle "**Datenschutz setzen**" und "**Datenschutz entfernen**" im Menü "**Bearbeiten**" (im Fenster der Variablendeklaration oder der E/A-Verdrahtung), um den individuellen Datenschutz des ausgewählten Objekts zu setzen oder zu entfernen. Beide Befehle erfordern die Eingabe eines gültigen Paßworts, damit die Zugriffsebene mit dem Objekt verknüpft werden kann. Daraufhin kann die individuell geschützten Variablen oder Kanalverdrahtungen nicht mehr ohne die Eingabe eines Paßworts einer ausreichend hohen Prioritätsebene modifiziert werden.

Achtung: Wenn eine Variable oder ein Kanal durch eine Zugriffsebene geschützt ist und das entsprechende Paßwort aus dem Datenschutzsystem entfernt wird und wenn kein höheres Paßwort definiert ist, kann die Variable oder die Kanalverdrahtung nicht mehr modifiziert werden, außer es wird ein neues Paßwort einer ausreichend hohen Zugriffsebene definiert.

A.26. Fortgeschrittene Programmiertechniken

Dieses Kapitel enthält zusätzliche Informationen zur ISaGRAF Workstation und dem ISaGRAF Zielsystem. Es ist Benutzern gewidmet, denen die ISaGRAF Werkzeuge und Methoden bereits geläufig sind.

A.26.1. Mehr über die ISaGRAF Werkzeuge

Wenn man die Bearbeitungswerkzeuge der ISaGRAF Workstation benutzt, kann man mit der **rechten Maustaste** ein Menü mit den hauptsächlichen Bearbeitungsbefehlen öffnen. Es wird am aktuellen Standort des Cursors angezeigt und reduziert die Mausaktionen bei Ausschneide- und Einfügebefehlen.
Die ISaGRAF Werkzeuge unterstützen die **Mehrfachausführung**. Ein Werkzeug kann zwar nicht zweimal geöffnet werden, um das gleiche Dokument zu bearbeiten, es ist jedoch möglich, mehrere Fenster mit dem gleichen Werkzeug zu öffnen und verschiedene Objekte parallel zu bearbeiten.
Andere Befehle sind verfügbar, um die Beschreibung der grafischen Schaltflächen in den Befehlsleisten anzuzeigen. Doppelklicken Sie auf eine unbenutzte Fläche in der Befehlsleiste, um den Inhalt der Befehlsleiste als Popup-Menü anzuzeigen. Verlagern Sie den Mauscursor auf eine grafische Schaltfläche, um den Befehl in Textform anzuzeigen.

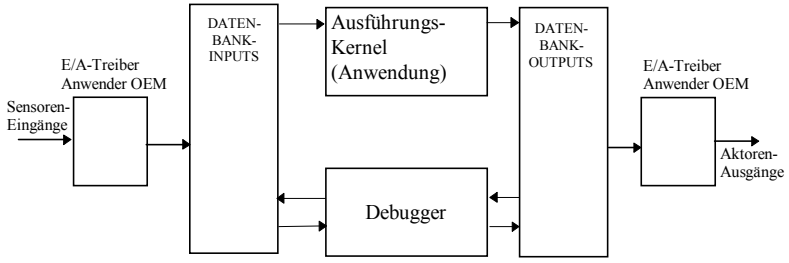
A.26.2. Verriegelte und virtuelle E/A

Wenn man eine E/A-Karte als **virtuell** definiert, wird die Verarbeitung der physikalischen E/A-Kanäle unterbrochen. Die Operationen des ISaGRAF Kernels verändern sich dagegen nicht. Der einzige Unterschied ist, daß die Eingänge der Sensoren nicht erfaßt und die Aktuatoren nicht aktualisiert werden. In diesem Modus kann der ISaGRAF Debugger benutzt werden, um die Eingangswerte zu modifizieren. Das Attribut **virtuell** betrifft eine komplette Karte. Es wird bei der Definition der E/A-Karte definiert, **vor** der Entwicklung des Anwendungscodes. Das Attribut **virtuell** ist eine **statische** Funktionalität und wird gespeichert, wenn die Anwendung gestoppt und wieder gestartet wird.
Zusätzlich besteht die Möglichkeit, die E/A-Variablen zu **verriegeln**. Darunter versteht man die Unterbrechung einer physikalischen Einrichtung und der entsprechenden ISaGRAF E/A-Variablen. Die Verriegelung und Freigabe einer Variablen erfolgt mit dem Debugger. Die Verriegelung einer Variablen ist eine **dynamische** Operation und wird beim Neustart der Anwendung nicht gespeichert. Eine **Verriegelung** bezieht sich jeweils auf **eine einzige** Variable (einen E/A-Kanal).

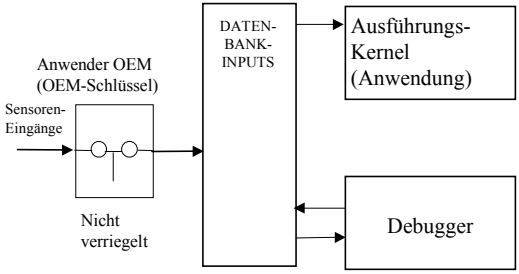
Es folgt eine Zusammenfassung der hauptsächlichen E/A-Steuerfunktionen:

	Attribut Virtuell	*Verriegelung*
Auswahlwerkzeug	E/A-Verdrahtung	Debugger
Definition	statisch	dynamisch
Auswahlmodus	Karte	Variable
Anwendung	Validation und Tests	Instandhaltung

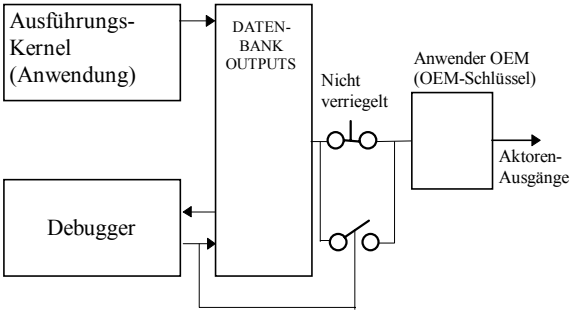
Das folgende Schema veranschaulicht den E/A-Datenfluß zwischen den ISaGRAF Tasks:



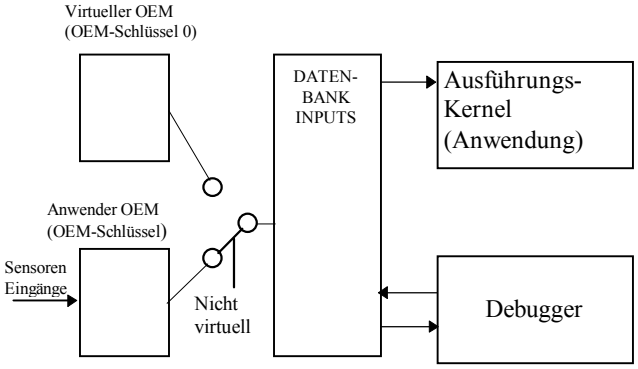
Wenn eine Variable verriegelt ist, bleiben die verschiedenen Zugriffe zur Datenbank unverändert, der Anschluß der Eingangseinrichtung ist jedoch unterbrochen. Die Eingangswerte können mit dem Debugger forciert und vom ISaGRAF Kernel verarbeitet werden:



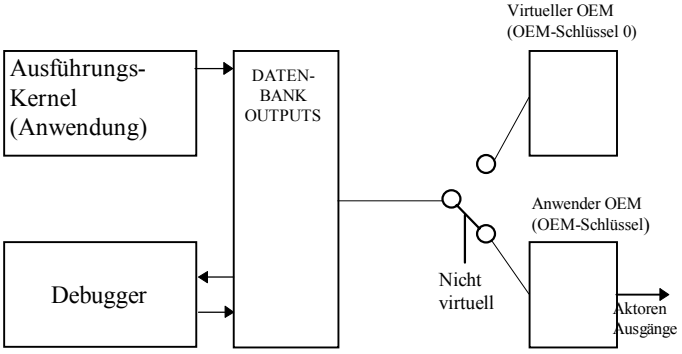
Wenn eine Ausgangsvariable verriegelt ist, ist der Anschluß des Ausführungskernels und des Ausgangstreibers unterbrochen. In diesem Fall kann noch mit Hilfe des Debuggers über den Ausgangstreiber auf die Ausgangseinrichtung zugegriffen werden:



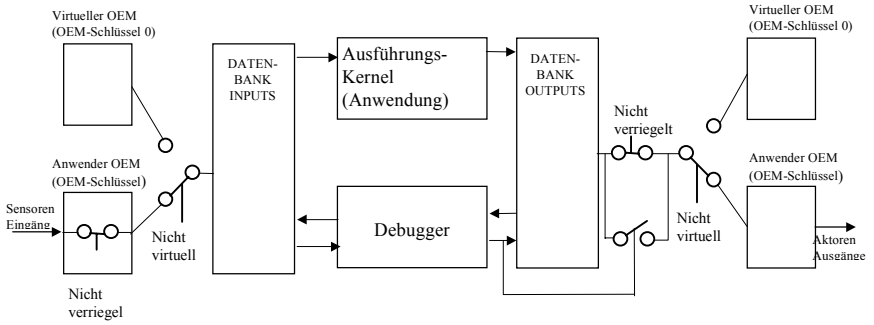
Wenn das Attribut "virtuell" für eine Eingangskarte definiert wurde, ist der Anschluß zur Datenbank der Eingänge und zu den zugeordneten Eingangseinrichtungen unterbrochen. Ein virtueller E/A-Treiber ersetzt den realen Treiber.



Die Definition einer virtuellen Ausgangskarte folgt denselben Regeln wie die einer virtuellen Eingangskarte. Bei virtuellen Ausgangskarten aktualisiert der ISaGRAF Kernel die Datenbank der Ausgänge. Der Anschluß zu dieser Datenbank und den zugeordneten Ausgangseinrichtungen ist jedoch unterbrochen. Ein virtueller E/A-Treiber ersetzt den realen Treiber.



Um alle Möglichkeiten zusammenzufassen:



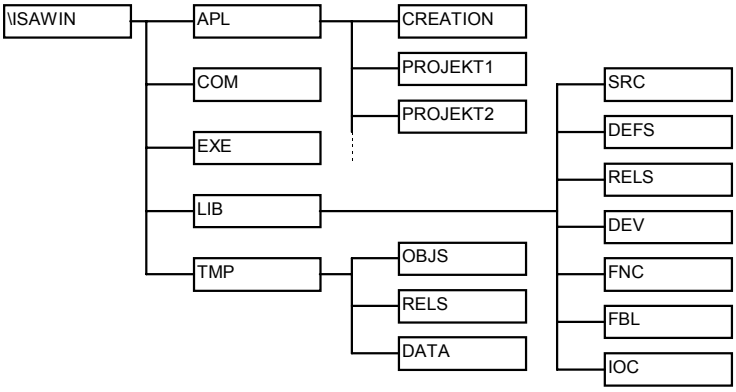
A.26.3. PC-SPS-Verbindung validieren

Die meisten Probleme, die durch eine schlechte Verbindung der ISaGRAF Workstation mit der Ziel-SPS entstehen, werden im Debugger-Fenster durch die Meldung "**Unterbrochen**" angezeigt. Vor einer Diagnose sollte die Kommunikation validiert werden, wenn in der Ziel-SPS **keine Anwendung aktiv ist**. Auf diese Weise kann die serielle Kommunikationsverbindung unabhängig von eventuellen Ausführungsproblemen validiert werden.
Die "C"-Sprache, die für die Beschreibung der Umrechnungsfunktionen und C-Funktionen benutzt wird, ermöglicht einen direkten Zugriff auf das Zielsystem. Ein Programmierfehler in einem derartigen Software-Komponenten kann Systemfehler oder ein fehlerhaftes Verhalten des ISaGRAF Systems bewirken. Diese Probleme können auch bei der Entwicklung von E/A-Treibern mit Hilfe der ISaGRAF **OEM**-Option entstehen. Es können beispielsweise Systemfehler auftreten, wenn eine E/A-Karte an einer ungültigen Busadresse implementiert wird. Die folgende Tabelle liefert eine Zusammenfassung der Fehlerdiagnose:

Status	*Kontext*	*Diagnose*
"Unterbrochen" (vor dem Laden)		- Zielsteuerung außer Betrieb - kein Kabel / ungültiges Kabel - ungültige Verbindungsparameter - ISaGRAF Zielsystem falsch installiert
"Unterbrochen" (nach dem Laden)	Start im Schrittmodus	- ungültige E/A-Konfiguration - Systemabsturz
	Start im Echtzeitmodus	- ungültige E/A-Konfiguration - Systemabsturz (durch "C"-Programmierung)
"Keine Anwendung"		- Anwendung nicht geladen - Anwendung nicht gestartet (durch "C"-Programmierung) - Intel/Motorola verwechselt - ungültige Version des Zielsystems

A.26.4. ISaGRAF Verzeichnisse

Die ISaGRAF Workstation arbeitet mit einer Struktur dedizierter Verzeichnisse. Das Wurzelverzeichnis dieser Architektur wird vom Benutzer bei der ISaGRAF Installation bestimmt. Der Vorgabename des ISaGRAF Wurzelverzeichnisses ist **ISAWIN**. Es folgt die vom Installationsprogramm auf der Festplatte erstellte Standard-Architektur:



Die Standard-Unterverzeichnisse von ISaGRAF sind:

VERZEICHNIS	*INHALT*
APL	Wurzelverzeichnis für die ISaGRAF Projekte. Jedes Projekt entspricht einem Unterverzeichnis, in dem alle Projektdaten enthalten sind. Es kann andere Verzeichnisse für andere Projektgruppen geben. Das ISaGRAF Installationsprogramm erstellt das Verzeichnis "**SMP**", in dem Beispielanwendungen gespeichert sind.
COM	Objekte mit "allgemeinem (common)" Geltungsbereich. Diese Daten können von allen Projekten benutzt werden.
EXE	ISaGRAF Programme und Hilfsdateien.
LIB	ISaGRAF Bibliotheken: - Elementlisten, - Parameter und Schnittstelle für jedes Element, - technische Datenblätter.
LIB\IOC	Quellcode für E/A-Konfigurationen
LIB\FNC	Quellcode für die in den IEC-Sprachen geschriebenen Funktionen
LIB\FBL	Quellcode für die in den IEC-Sprachen geschriebenen Funktionsbausteine
LIB\SRC	Quellcode für Umrechnungen und C-Funktionen
LIB\DEFS	Quellcode-Vorsatz für Umrechnungen und C-Funktionen
LIB\RELS	Objektcode der Umrechnungen und C-Funktionen
LIB\DEV	Befehlsdateien für die Entwicklung von "C"-Bibliotheken: Makefiles, Verknüpfungslisten, etc...

TMP	Temporäre Dateien: Die Unterverzeichnisse von TMP sind dem ISaGRAF Codegenerator vorbehalten und können nicht gelöscht werden.

Die Unterverzeichnisse können auf der Festplatte an eine andere Stelle verschoben werden. Wenn eine nicht-standard Architektur benutzt wird, müssen die Pfadnamen der Unterverzeichnisse im Abschnitt **WS00**, in der Initialisierungsdatei **ISA.ini**, im ISaGRAF Unterverzeichnis **EXE**, deklariert werden. Es folgen die Eingaben des Abschnitts **WS001**:

Isa	Wurzelverzeichnis für die ISaGRAF Architektur
IsaExe	Wurzelverzeichnis für die ISaGRAF Programme und Hilfsdateien
IsaApl	Wurzelverzeichnis für die ISaGRAF Projekte
IsaTmp	Verzeichnis für die temporären Dateien
IsaSrc	Verzeichnis für die Quellcode-Bibliothek
IsaDefs	Verzeichnis für die Quellcode-Vorsätze der Bibliotheken

Achtung: Wenn Sie IsaTmp in ein anderes Verzeichnis verschieben, müssen Sie die Unterverzeichnisse OBJS, RELS und DATA im neuen Verzeichnis erstellen. Das folgende Beispiel veranschaulicht die Eingaben des Abschnitts **WS001** für eine Neudefinition der ISaGRAF Standardarchitektur:

```
;datei c:\ISAWIN\EXE\ISA.ini

[WS001]
Isa=c:\isawin
IsaExe=c:\isawin\exe
IsaApl=c:\isawin\apl
IsaTmp=c:\isawin\tmp
IsaSrc=c:\isawin\lib\src
IsaDefs=c:\isawin\lib\defs
```

Wenn Sie dem ISaGRAF Zielsystem C-Funktionen und -Funktionsbausteine hinzufügen möchten, wird das Verzeichnis **\ISAWIN\LIB\DEV** benutzt, um Entwicklungsdateien zu speichern: Befehlsdateien, Makefiles, Maps, etc... Das Verzeichnis **\ISAWIN\LIB\RELS** wird benutzt, um die bei der "C"-Kompilierung generierten Objektdateien, sowie die ISaGRAF "C"-Bibliotheken, die für die LINK-Operationen gebraucht werden, zu speichern.

A.26.5. Anwendungssymbole

Sämtliche Objekte einer ISaGRAF Anwendung sind mit jeweils einem Namen, der bei der Variablendeklaration eingegeben wird, und einer vom Codegenerator berechneten, internen **virtuellen** Adresse gekennzeichnet. Die virtuelle Adresse einer Variablen darf nicht mit ihrer **Netzwerkadresse**, die ebenfalls bei der Variablendeklaration eingegeben wird, verwechselt werden. Virtuelle Adressen werden für Kommunikationsvorgänge und spezielle "C"-Anwendungen, die die OEM-Option benutzen, verwendet. Wenn der ISaGRAF Codegenerator gestartet wird, generiert er eine ASCII-Datei mit der logischen Entsprechung zwischen den Namen und virtuellen Adressen sämtlicher Objekte (Variablen, Programme, Schritte...) des betreffenden Projekts. Diese Datei kann problemlos befragt werden,

wenn Informationen über die statische ISaGRAF Datenbank benötigt werden. Die Datei heißt "**APPLI.TST**" und befindet sich im Verzeichnis des ISaGRAF Projekts: "**\ISAWIN\APL\proname**" (proname ist der Name des Projekts). Der folgende Abschnitt beschreibt das Format der Datei "**APPLI.TST**". Die hauptsächlichen Notationen für die Beschreibung sind:

VA	virtuelle Adresse
ATTR	Attribut einer Variablen
USP	"C"-Funktion

Hier die möglichen Werte für die Attribute einer Variablen. Diese Werte erscheinen im Feld "**Attribut**":

+X	interne Variable
+C	interne nur-lesen Variable
+I	Eingangsvariable
+O	Ausgangsvariable

Alle Zahlen, mit Ausnahme der virtuellen Adressen, werden als ganzzahlige Dezimalwerte ausgedrückt. Die virtuellen Adressen (**VA**) werden als vierstellige Hexadezimalzahlen mit dem Vorzeichen "**!**" ausgedrückt. Zum Beispiel:

`123`	ist ein Dezimalwert
`!A003`	ist eine virtuelle Hexadezimaladresse

Hier das globale Format der Datei "**APPLI.TST**". Die Datei besteht aus einer Liste von **Blöcken**. Ein Block besteht aus einer Folge von **Datensätzen**. Jeder Datensatz wird auf einer Textzeile beschrieben und beginnt mit einer Kopfzeile.

```
Anfangsblock
Beschreibungsblöcke
Endblock
```

Es folgt die allgemeine Syntax eines Blocks:

```
@ <blockname> <argumente>
#datensatz...
#datensatz...
...
```

Es folgt die Struktur des ersten Blocks, der die hauptsächlichen Informationen über die Anwendung enthält:

```
@ISA_SYMBOLS,<appli_crc>
#NAME,<appli_name>,<version>
#DATE,<creation_date>
#SIZE,G=<nbprg>,S=<nbstep>,T=<nbtra>,L=0,P=<nbpro>,V=<nbvar>
#COMMENT,cj international
```

appli_crc	Checksum der Anwendungsymbole
appli_name	Name der Anwendung

version	Versionsnummer der ISaGRAF Workstation
creation_date	Entwicklungsdatum der Anwendung
nbprg	Anzahl der Programme
nbstep	Anzahl der AS-Schritte
nbtra	Anzahl der AS-Transitionen
nbpro	Anzahl der verwendeten "C"-Funktionen
nbvar	Anzahl sämtlicher Variablen

Hier die Struktur des Endblocks, der das Ende der Datei kennzeichnet:

```
@END_SYMBOLS
```

Es folgt die Struktur des Blocks, der für die Beschreibung der Anwendungsprogramme benutzt wird:

```
@PROGRAMS,<nbprg>
#<va>,<name>
#...
```

nbprg	Anzahl der in diesem Block definierten Programme
va	virtuelle Adresse des Programms
name	Programmname

Es folgt die Struktur des Blocks, der für die Beschreibung der AS-Schritte der Anwendung benutzt wird. Für jedes nicht-AS-Programm wird ein virtueller Schritt definiert:

```
@STEPS,<nbsteps>
#<va>,<name>,<father>
#...
```

nbsteps	Anzahl der in diesem Block definierten Schritte
va	virtuelle Adresse des Schritts
name	Name des Schritts
father	virtuelle Adresse des Vaterprogramms

Es folgt die Struktur des Blocks, der für die Beschreibung der AS-Transitionen der Anwendung benutzt wird:

```
@TRANSITIONS,<nbtrans>
#<va>,<name>,<father>
#...
```

nbtrans	Anzahl der in diesem Block definierten Transitionen
va	virtuelle Adresse der Transition
name	Name der Transition
father	virtuelle Adresse des Vaterprogramms

Es folgt die Struktur des Blocks, der für die Beschreibung der Booleschen Variablen der Anwendung benutzt wird:

```
@BOOLEANS,<nb_boo>
```

```
#<va>,<name>,<attr>,<program>,<eq_false>,<eq_true>
#...
```

und wenn die Anzahl der Variablen 4095 überschreitet:

```
X#(1.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_boo	Anzahl der Variablen in diesem Block
va	virtuelle Adresse der Variablen
varno	Adressenbereich (innerhalb des Booleschen Datentyps)
name	Name der Variablen
attr	Attribut der Variablen
program	virtuelle Adresse des Vaterprogramms oder "**!0000**" für eine globale Variable
eq_false	Zeichenkette für einen False-Wert
eq_true	**false** Zeichenkette für einen True-Wert

Es folgt die Struktur des Blocks, der für die Beschreibung der analogen Variablen der Anwendung benutzt wird:

```
@ANALOGS,<nb_ana>
#<va>,<name>,<attr>,<program>,<format>,<unit>
#...
```

und wenn die Anzahl der Variablen 4095 überschreitet:

```
X#(2.<varno>),<name>,<attr>,<program>,<format>,<unit>
```

nb_ana	Anzahl der Variablen in diesem Block
va	virtuelle Adresse der Variablen
varno	Adressenbereich (innerhalb des analogen Datentyps)
name	Name der Variablen
attr	Attribut der Variablen
program	virtuelle Adresse des Vaterprogramms oder "**!0000**" für eine globale Variable
format	= "**I**" für eine ganzzahlige Variable = **"F**" für eine reale Variable
unit	Einheit-Zeichenkette

Es folgt die Struktur des Blocks, der für die Beschreibung der Timer-Variablen der Anwendung benutzt wird:

```
@TIMERS,<nb_tmr>
#<va>,<name>,<attr>,<program>
#...
```

und wenn die Anzahl der Variablen 4095 überschreitet:

```
X#(3.<varno>),<name>,<attr>,<program>
```

nb_tmr	Anzahl der Variablen in diesem Block
va	virtuelle Adresse der Variablen

varno	Adressenbereich (innerhalb des Timer-Datentyps)
name	Name der Variablen
attr	Attribut der Variablen (immer +X: intern)
program	virtuelle Adresse des Vaterprogramms oder "**!0000**" für eine globale Variable

Es folgt die Struktur des Blocks, der für die Beschreibung der Zeichenketten-Variablen der Anwendung benutzt wird:

```
@MESSAGES,<nb_msg>
#<va>,<name>,<attr>,<program>,<max_len>
#...
```

und wenn die Anzahl der Variablen 4095 überschreitet:

```
X#(4.<varno>),<name>,<attr>,<program>,<max_len>
```

nb_msg	Anzahl der Variablen in diesem Block
va	virtuelle Adresse der Variablen
varno	Adressenbereich (innerhalb des Zeichenketten-Datentyps)
name	Name der Variablen
attr	Attribut der Variablen
program	virtuelle Adresse des Vaterprogramms oder "**!0000**" für eine globale Variable
max_len	maximale Länge (deklarierte Kapazität)

Es folgt die Struktur des Blocks, der für die Beschreibung der "C"-Funktionen der Anwendung benutzt wird:

```
@USP,<nb_usp>
#<va>,<name>
#...
```

nb_usp	Anzahl an C-Funktionen in diesem Block
va	virtuelle Adresse der C-Funktion
name	Name der C-Funktion

Es folgt die Struktur des Blocks, der für die Beschreibung der "C"-Funktionsbaustein-Instancen der Anwendung benutzt wird:

```
@FBINSTANCES,<nb_fb>
#<va>,<inst_name>,<fb_name>
#...
```

nb_fb	Anzahl der Instancen eines C-Funktionsbausteins in diesem Block
va	virtuelle Adresse der C-Funktionsbaustein-Instance
inst_name	Name der C-Funktionsbaustein-Instance
fb_name	Referenzname des C-Funktionsbausteins

A.26.6. Begrenzungen der ISaGRAF Workstation "LARGE" (WDL)

Die folgenden Begrenzungen gelten für die Objekte der ISaGRAF Workstation. Durch die Konfiguration Ihres Computers (Arbeitsspeicher, Speicherplatz auf der Festplatte) und die Kapazitäten des ISaGRAF Zielsystems (Arbeitsspeicher, verfügbare Hardware- und Software-Ressourcen) können natürlich noch weitere praktische Einschränkungen entstehen. In den folgenden Tabellen werden die äußersten Grenzwerte aufgeführt, die auf keinen Fall überschritten werden dürfen.

Für ein Projekt:

Objekt	*Maximum*	*Anmerkungen*
Programme	255	für alle Haupt-, Sohn- und Unterprogramme
Hierarchie-Ebenen	20	

Die Anzahl der in der Workstation installierten Projekte wird nur durch den verfügbaren Speicherplatz auf der Festplatte begrenzt.

Namen:

Name für:	*Maximum*	*Anmerkungen*
Projekt	8 Zeichen	
Programm	8 Zeichen	
Variable	16 Zeichen	+ 60 Zeichen für den Kommentar
Definition	16 Zeichen	
Definierte Äquivalenz	255 Zeichen	+ 60 Zeichen für den Kommentar
Umrechnungstabelle	16 Zeichen	
Variablenliste	16 Zeichen	
Funktion / FB (Bibl.)	8 Zeichen	betrifft C-Funktionen, C-Funktionsbausteine oder in einer IEC-Sprache geschriebene Funktionen
Funktionsparameter	16 Zeichen	betrifft C-Funktionen, C-FB oder in einer IEC-Sprache geschriebene Funktionen
E/A-Karte	8 Zeichen	
E/A-Konfiguration	8 Zeichen	
OEM-Parameter (Karte)	16 Zeichen	
Umrechnungsfunktion	8 Zeichen	

Bearbeitung (für ein Programm):

Objekt	*Maximum*	*Anmerkungen*
AS-Zeilen	600	
AS-Spalten	20	
AS-Schritte	4095	für das gesamte Projekt (einschließlich der Schritte, Initialisierungsschritte, Anfangs- und Endschritte)
AS-Transitionen	4095	für die gesamte Anwendung
KOP/FBS-Bearbeitung	200 Spalten 2000 Zeilen	Dies ist die Größe des Bearbeitungsfelds (Zellen).
Schneller KOP	unbegrenzt	entsprechend der Kapazität

		des benutzten Computers
AWL-Marken	251	in einem AWL-Programm
Textbearbeitung	40KBytes	

Datenverzeichnis (für ein Projekt):

Objekt	*Maximum*	*Anmerkungen*
Boolesche Variablen	65535	
Analoge Variablen	65535	ganzzahlige und reale Variablen zusammengenommen
Timer-Variablen	65535	
Zeichenketten-Variablen	65535	
Definitionen	4095	in einer Liste (gleicher Geltungsbereich)
Definitionen	255	in einem Programm
Umrechnungstabellen	127	in einer Anwendung
Punkte in einer Tabelle	32	in einer Umrechnungstabelle

Die Begrenzungen für die Anzahl der Booleschen, analogen und Zeichenketten-Variablen gelten für interne, Eingangs- und Ausgangsvariablen zusammengenommen.

E/A-Verdahtung:

Objekt	*Maximum*	*Anmerkungen*
E/A-Karten	256	in einer Anwendung (Karten oder komplexe Baugruppen)
E/A-Kanäle	128	auf einer Karte

Die Anzahl der E/A-Karten, einschließlich einzelner Karten und Elemente komplexer Baugruppen, darf 256 nicht überschreiten.

Bibliotheken:

Objekt	*Maximum*	*Anmerkungen*
Funktionen (IEC)	255	zusammen in der Bibliothek installiert
F-Bausteine (IEC)	255	zusammen in der Bibliothek installiert
C-Funktionen	255	zusammen in der Bibliothek installiert
C-Funktionsbausteine	255	zusammen in der Bibliothek installiert
FB-Instancen	4095	für den gleichen Funktionsbausteintyp in einer Anwendung
F-Eingangsparameter	31	betrifft C-Funktionen und Funktion in den IEC-Sprachen
FB-Parameter	32	Ein- und Ausgangsparameter zusammen. Es ist mindestens ein Ausgangsparameter erforderlich.
Umrechnungsfunktionen	128	zusammen in der Bibliothek installiert
E/A-Konfigurationen	255	zusammen in der Bibliothek installiert
E/A-Karten	255	zusammen in der Bibliothek installiert
E/A-Baugruppen.	255	zusammen in der Bibliothek installiert
OEM-Kartenparameter	16	

B. Sprachreferenzen

B.1. Projektarchitektur

Ein ISaGRAF Projekt besteht aus mehreren Programmiereinheiten, die **Programme** genannt werden. Diese Projektprogramme sind in einer hierarchischen Architektur angeordnet. Sie können in den Grafik- oder Textsprachen **AS**, **FD (Flußdiagramm), FBS**, **KOP, ST** oder **AWL** geschrieben werden.

B.1.1. Programme

Ein **Programm** ist eine logische Programmiereinheit, die Operationen zwischen den **Variablen** eines Prozesses beschreibt. Dabei handelt es sich um **sequentielle** oder **zyklische** Operationen. Zyklische Programme werden automatisch bei jedem Zielsystemzyklus ausgeführt. Die sequentielle Programmausführung folgt den dynamischen Regeln der **AS-** oder **FD-Sprache**.
Die Programme sind in einer hierarchischen Baumstruktur angeordnet. Programme, die sich auf der höchsten Prioritätsebene befinden, werden vom System aktiviert. Die Unterprogramme (auf niedrigeren Prioritätsebenen) werden von ihren Vaterprogrammen gesteuert. Die Programmbeschreibungen erfolgen mit Hilfe der folgenden Grafik- oder Textsprachen :
- **Ablaufsprache** (AS) für die Programmierung hoher Prioritätsebenen
- **Flußdiagramm** (FD) für die Programmierung hoher Prioritätsebenen
- **Funktionsbaustein-Sprache** (FBS) für komplexe zyklische Operationen
- **Kontaktplan** (KOP) nur für boolesche Operationen
- **Strukturierter Text** (ST) für beliebige zyklische Operationen
- **Anweisungsliste** (AWL) für Operationen niedriger Ebenen

Für die Programmierung eines Programms können nicht mehrere Sprachen benutzt werden, mit Ausnahme von KOP und FBS, die in einem Diagramm miteinander kombiniert werden können.

B.1.2. Zyklische und sequentielle Operationen

Die Programmhierarchie besteht aus vier hauptsächlichen **Abschnitten** oder Gruppen :

ANFANG	Programme, die am Anfang eines jeden Zielsystemzyklus ausgeführt werden
SEQUENTIELL	Programme, die den dynamischen AS- oder FD-Regeln folgen
ENDE	Programme, die am Ende eines jeden Zielsystemzyklus ausgeführt werden
FUNKTIONEN	ein Satz nicht-dedizierter Unterprogramme

Die Programme der Abschnitte "**Anfang**" und "**Ende**" beschreiben zyklische Operationen und sind zeitunabhängig. Die Programme des Abschnitts "**Sequentiell**" beschreiben sequentielle Operationen, wobei die Basisoperationen expliziert von der Timer-Variablen synchronisiert werden. Die Hauptprogramme des Abschnitts "**Anfang**" werden systematisch am Anfang eines jeden Zyklus, die Hauptprogramme des Abschnitts "**Ende**" systematisch am Ende eines jeden Zyklus

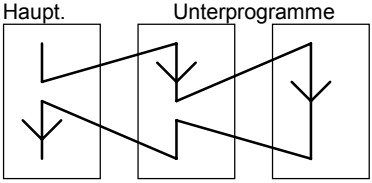
ausgeführt. Die Hauptprogramme des Abschnitts "**Sequentiell**" folgen den dynamischen **AS**- oder **FD**-Regeln.
Programme des Abschnitts "**Funktionen**" sind Unterprogramme, die von allen anderen Programmen des Projekts aufgerufen werden können. Ein Programm des Abschnitts "**Funktionen**" kann ein Unterprogramm dieses Abschnitts aufrufen.
Die Haupt- und Sohnprogramme des sequentiellen Abschnitts werden in der **AS**- oder **FD**-Sprache geschrieben, im Gegensatz zu den Programmen der zyklischen Abschnitte (**Anfang** und **Ende**), die nicht in der **AS**- oder **FD**-Sprache geschrieben werden können. Jedes Programm eines jeden Abschnitts kann ein oder mehrere Unterprogramme besitzen. Ein Programm des sequentiellen Abschnitts kann ein oder mehrere **AS**- oder **FD**-Sohnprogramme (deren Sprache seiner eigenen Programmiersprache entspricht) steuern. Unterprogramme können nicht in der **AS**- oder **FD**-Sprache geschrieben werden.
Die Programme des Abschnitts "**Anfang**" werden benutzt, um einleitende, auf den Eingangsvariablen ausgeführte Operationen zu beschreiben, um höhere gefilterte Variablen zu konstruieren. Solche Variablen werden häufig von den Programmen des Abschnitts "**Sequentiell**" verwendet. Die Programme des Abschnitts "**Ende**" werden benutzt, um Sicherheitsoperationen für Variablen zu beschreiben, die im Abschnitt "**Sequentiell**" benutzt werden, bevor die Werte zu den Ausgangseinrichtungen gesendet werden.

B.1.3. AS- und FD-Sohnprogramme

Ein beliebiges **AS**-Programm des sequentiellen Abschnitts kann andere **AS**-Programme, die **AS-Sohnprogramme** genannt werden, steuern. Ein **AS-Sohnprogramm** ist ein parallel ausgeführtes Programm, das von seinem Vaterprogramm gestartet, gestoppt, angehalten oder erneut gestartet werden kann. Das Vaterprogramm sowie sein Sohnprogramme werden beide in der **AS-**Sprache geschrieben. AS-Sohnprogramme können lokale Variablen und Definitionen besitzen.
Wenn ein Vaterprogramm ein **AS**-Sohnprogramm startet, setzt es eine AS-**Zugriffsberechtigung** (Aktivierung) in jeden seiner Initialisierungsschritte. Dieser Befehl wird mit der **GSTART**-Anweisung geschrieben. Wenn ein Vaterprogramm ein **AS**-Sohnprogramm stoppt, entfernt es alle vorhandenen Zugriffsberechtigungen aus seinen **Schritten**. Dieser Befehl wird mit der **GKILL-**Anweisung geschrieben.
Wenn ein Vaterprogramm ein **AS**-Sohnprogramm anhält, entfernt es alle vorhandenen Zugriffsberechtigungen aus seinen Schritten und speichert deren Position. Dieser Befehl wird mit der **GFREEZE-**Anweisung geschrieben. Wenn ein Vaterprogramm ein angehaltenes **AS**-Sohnprogramm erneut startet, ersetzt es alle beim Anhalten entfernten Zugriffsberechtigungen in den Schritten. Dieser Befehl wird mit der **GRST-**Anweisung geschrieben.
Ein beliebiges **FD**-Programm des sequentiellen Abschnitts kann andere **FD**-Unterprogramme steuern. Das FD-Vaterprogramm wird blockiert (wartet), bis die Ausführung des FD-Unterprogramms beendet ist. Es ist nicht möglich, gleichzeitige Operationen in einem FD-Vaterprogramm und einem seiner FD-Unterprogramme auszuführen.

B.1.4. Funktionen und Unterprogramme

Die Ausführung eines Unterprogramms oder einer Funktion wird von seinem Vaterprogramm gesteuert. Das Vaterprogramm wird solange unterbrochen, bis die Ausführung des Unterprogramms oder der Funktion vollendet ist :



Jedes Programm eines jeden Abschnitts kann ein oder mehrere Unterprogramme besitzen. Ein Unterprogramm kann nur von einem einzigen Vaterprogramm aktiviert werden. Unterprogramme können lokale Variablen und Definitionen besitzen. Für die Beschreibung eines Unterprogramms können alle Sprachen, mit Ausnahme der **AS**- und **FD**-Sprachen, verwendet werden. Die Programme des Abschnitts "**Funktionen**" sind Unterprogramme, die von beliebigen Programmen des Projekts aufgerufen werden können. Im Gegensatz zu anderen Unterprogrammen sind sie keinem Vaterprogramm dediziert. Ein Programm des Abschnitts "**Funktionen**" kann ein Unterprogramm dieses Abschnitts aufrufen.

Achtung: Das ISaGRAF System unterstützt keine **Rekursivität** in den Funktionsaufrufen. Wenn ein Programm des Abschnitts "**Funktionen**" von sich selbst oder von einem seiner untergeordneten Programme aufgerufen wird, entsteht ein Ausführungsfehler.

Achtung: Eine Funktion oder ein Unterprogramm kann die lokalen Werte seiner lokalen Variablen nicht "speichern". Eine Funktion oder ein Unterprogramm ist nicht instanziiert, es kann also keine Funktionsbausteine aufrufen.

Die Schnittstelle eines Unterprogramms muß ausdrücklich definiert werden, mit einem **Typ** und einem **einmaligen Namen** für jeden seiner Aufrufparameter und den Returnparameter. Gemäß der Schreibkonventionen der **ST**-Sprache muß der Returnparameter eines Unterprogramms den gleichen Namen wie das Unterprogramm tragen.

Die folgende Tabelle zeigt, wie der Returnparameter im Rumpf eines Unterprogramms in den verschiedenen Sprachen forciert wird:

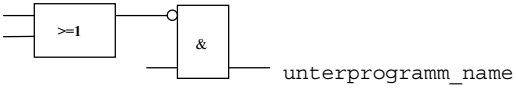
ST: Unter Anwendung seines Namens (der gleiche Name wie das Unterprogramm) wird dem Returnparameter ein Wert zugewiesen.

```
unterprogramm_name:= <ausdruck>;
```

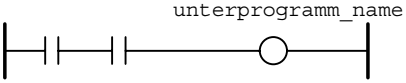
AWL: Der Wert des aktuellen Ergebnisses (AWL-Verzeichnis) am Ende der Sequenz wird im Returnparameter gespeichert:

```
LD      10
ADD     20        (* Wert des Returnparameters = 30 *)
```

FBS: Zuordnung des Returnparameters unter Anwendung seines Namens:

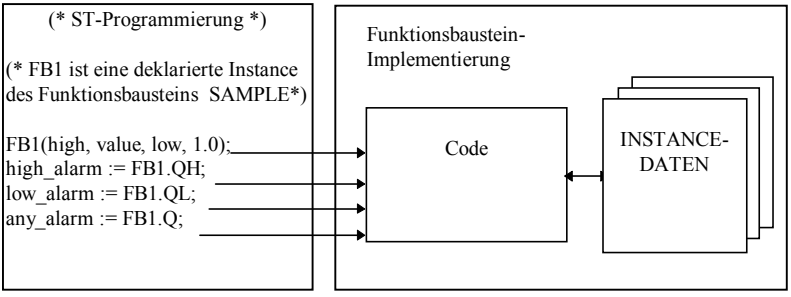


KOP: Benutzen Sie ein Spulensymbol mit dem Namen des Unterprogramms:



B.1.5. Funktionsbausteine

Funktionsbausteine können die Sprachen KOP, FBS, ST oder AWL benutzen. Funktionsbausteine sind instanziiert. Dies bedeutet, daß die lokalen Variablen eines Funktionsbausteins für jede Instance kopiert werden. Wenn ein Baustein in einem Programm aufgerufen wird, wird in Wirklichkeit die Instance des Baustein aufgerufen: es wird zwar der gleiche Code aufgerufen, aber die benutzten Daten sind die, welche der Instance zugeordnet wurden. Die Variablenwerte der Instance werden von einem Zyklus zum anderen gespeichert.



Achtung:

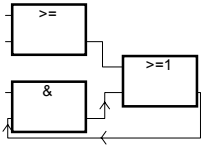
- Ein Funktionsbaustein, der in einer der IEC-Sprachen geschrieben wurde, kann keine anderen Funktionsbausteine aufrufen: Der Instanziierungsmechanismus verwaltet nur die lokalen Variablen des Bausteins selbst. Die folgenden Standardfunktionsbausteine können nicht in einem IEC-Funktionsbaustein benutzt werden:
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALRM, INTEGRAL, DERIVATE, BLINK, SIG_GEN

- Aus dem gleichen Grund können auch keine positiven oder negativen Kontakte und Spulen oder Setze- und Rücksetze-Spulen benutzt werden.

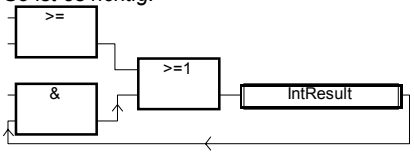
- Die Funktionen TSTART und TSTOP, die dazu dienen, Timer zu starten und zu stoppen, können nicht in Funktionsbausteinen für 3.0x Zielsysteme benutzt werden. Ihre Benutzung ist erst ab dem Zielsystem 3.20 möglich.

- Wenn eine Schleife in einem Funktionsbaustein benötigt wird, muß vor der Schleife eine lokale Variable eingesetzt werden. Es folgt ein Beispiel:

Diese Schleife funktioniert nicht:



So ist es richtig:



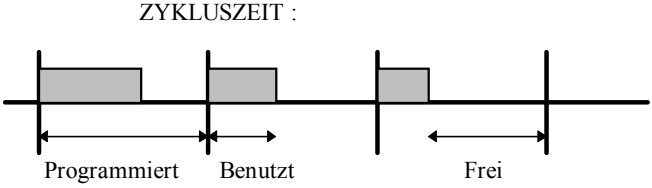
B.1.6. Programmiersprache

Ein Programm kann in einer der folgenden Grafik- oder Textsprachen geschrieben werden:
- **Ablaufsprache** (AS) für die Programmierung hoher Prioritätsebenen
- **Flußdiagramm** (FD) für die Programmierung hoher Prioritätsebenen
- **Funktionsbaustein-Sprache** (FBS) für komplexe zyklische Operationen
- **Kontaktplan** (KOP) nur für boolesche Operationen
- **Strukturierter Text** (ST) für beliebige zyklische Operationen
- **Anweisungsliste** (AWL) für Operationen niedriger Ebenen

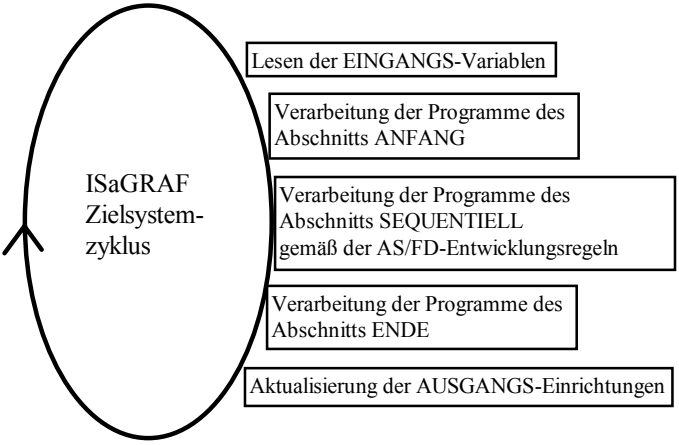
Für die Programmierung eines Programms können nicht mehrere Sprachen benutzen werden, mit Ausnahme von KOP und FBS, die in einem Diagramm miteinander kombiniert werden können. Die Programmiersprache wird bei der Programmerstellung gewählt und kann später nicht mehr geändert werden

B.1.7. Ausführungsregeln

ISaGRAF ist ein **synchrones** System. Sämtliche Operationen sind zeitabhängig. Die Dauer einer Zeitspanne wird **Zykluszeit** genannt :



In einem Zielsystemzyklus erfolgen die folgenden Grundoperationen:



Dieses System garantiert:
- daß die Eingangsvariablen in einem Zyklus unverändert bleiben,
- daß die Ausgangseinrichtungen nicht öfter als einmal pro Zyklus aktualisiert werden,
- daß verschiedene Programme zuverlässig mit einer globalen Variablen arbeiten können,
- daß die Antwortzeit der gesamten Anwendung geschätzt und gesteuert werden kann.

B.2. Gemeinsame Objekte

In diesem Kapitel werden die **Objekte** der ISaGRAF Programmierungs-Datenbank vorgestellt. Sie werden von Programmen manipuliert, die in den **AS-**, **FD-**, **FBS-**, **KOP-**, **ST-** und **AWL-**Sprachen geschrieben werden.

B.2.1. Basistypen

Alle konstanten oder variablen Ausdrücke, die in einem Programm benutzt werden, müssen mit einem Typ gekennzeichnet werden. Die Typ-Konsistenz muß in den Netzwerken und in Textausdrücken respektiert werden. Hier die ISaGRAF Basistypen :

	logischer Wert (true oder false)
ANALOG:	kontinuierlicher, ganzzahliger oder realer Wert
TIMER:	Zeitwert
ZEICHENKETTE:	Zeichenfolge

Anmerkung : Der in einem Timer enthaltene Wert ist stets kleiner als ein Tag und darf nicht für ein Datum benutzt werden.

B.2.2. Konstante Ausdrücke

Konstante Ausdrücke sind stets typ-bezogen, d. h. daß eine Notation nicht für konstante Ausdrücke unterschiedlichen Typs verwendet werden darf.

B.2.2.1. Konstante boolesche Ausdrücke

Es gibt lediglich zwei konstante Ausdrücke vom Typ boolesch :

TRUE	ist äquivalent zu dem ganzzahligen Wert 1
FALSE	ist äquivalent zu dem ganzzahligen Wert 0

Bei der Eingabe der Schlüsselwörter "True" und "False" wird zwischen Groß- und Kleinschreibung nicht unterschieden.

B.2.2.2. Konstante ganzzahlige analoge Ausdrücke

Ein konstanter ganzzahliger analoger Ausdruck ist eine vorzeichenbehaftete Zahl zwischen **-2147483647** und **+2147483647**. Ganzzahlige, konstante Ausdrücke können in einer der folgenden Grundgrößen ausgedrückt werden. Konstante Ausdrücke müssen mit einem **Vorzeichen** beginnen, das deren Grundgröße kennzeichnet.

Grundgröße	Vorzeichen	Beispiel
DEZIMAL	(kein)	-908
HEXADEZIMAL	"16#"	16#1A2B3C4D
OKTAL	"8#"	8#1756402

BINÄR	"2#"	2#1101_0001_0101_1101

Das Unterstrich-Zeichen ('**_**'), das keine semantische Bedeutung hat, kann beliebig zum Trennen einzelner Zeichengruppen verwendet werden und steigert die Leserlichkeit längerer Ausdrücke.

B.2.2.3. Konstante reale analoge Ausdrücke

Reale, analoge Ausdrücke können **dezimal** oder **wissenschaftlich** dargestellt werden. Ein **Dezimalpunkt** ('**.**') trennt den dezimalen vom ganzen Zahlenteil. Dieser Punkt ist absolut notwendig, um ganzzahlige, konstante Ausdrücke von den realen Konstanten unterscheiden zu können. In der wissenschaftlichen Darstellungsweise werden die Buchstaben '**E**' oder '**F**' benutzt, um die **Mantisse** vom **Exponenten** zu trennen. Der Exponent eines wissenschaftlichen, konstanten Ausdrucks ist eine vorzeichenbehaftete Ganzzahl zwischen **-37** und **+37**. Hier einige Beispiele konstanter realer Ausdrücke :

3.14159 -1.0E+12
+1.0 1.0F-15
-789.56 +1.0E-37

Der Ausdruck "**123**" stellt keinen realen Wert dar. Diese Notation schreibt man richtig "**123.0**".

B.2.2.4. Timer-Konstanten

Konstante Ausdrücke vom Typ Timer sind Zeitwerte zwischen **0 Sekunden** und **23h59m59s99ms**. Die kleinste, zugelassene Einheit ist eine Millisekunde. Die folgenden Standard-Zeiteinheiten werden in konstanten Ausdrücken vom Typ Timer benutzt:

Stunde:	Der Buchstabe "**h**" folgt auf die Anzahl der Stunden
Minute:	Der Buchstabe "**m**" folgt auf die Anzahl der Minuten
Sekunde:	Der Buchstabe "**s**" folgt auf die Anzahl der Sekunden
Millisekunde:	Die Buchstaben "**ms**" folgen auf die Anzahl der Millisekunden

Der Ausdruck muß mit dem Vorzeichen "**T#**" oder "**TIME#**" beginnen. Für die Vorzeichen oder Buchstaben sind Groß- und Kleinbuchstaben zulässig. Bestimmte Einheiten können weggelassen werden. Hier einige Beispiele für konstante Ausdücke vom Typ Timer :

T#1H451MS	1 Stunde und 451 Millisekunden
time#1H3M	1 Stunde und 3 Minuten

Der Ausdruck "0" ist keine Timer-Konstante, sondern ein analoger Ausdruck.

B.2.2.5. Konstante Ausdrücke vom Typ Zeichenkette

Konstante Ausdrücke vom Typ Zeichenkette sind Zeichenfolgen, die zwischen zwei Apostrophen geschrieben werden. Zum Beispiel :

'DIES IST EINE ZEICHENKETTE'

Achtung: Das Apostroph-Zeichen darf nicht innerhalb der Zeichenkette verwendet werden. Ein konstanter Ausdruck vom Typ Zeichenkette muß auf einer einzigen Programmzeile geschrieben werden und darf 255 Zeichen nicht überschreiten.
Eine leere Zeichenkette wird durch zwei Apostroph-Zeichen gekennzeichnet, zwischen denen keine Leerzeichen oder Tabs erlaubt sind:

'' (* dies ist eine leere Zeichenkette *)

Das Spezialzeichen Dollar ('**$**'), gefolgt von anderen Spezialzeichen, kann benutzt werden, um nicht ausdruckbare Zeichen in eine Zeichenkette einzufügen :

Sequenz	Bedeutung	Ascii	Beispiel
$$	'$'-Zeichen	16#24	'Der $$ steht...'
$'	Apostroph	16#27	'Geben Sie $'O$' für YES ein '
$L	folgende Zeile	16#0a	'folgende $L Zeile'
$R	Wagenrückgang	16#0d	' Tag $R guten'
$N	neue Zeile	16#0d0a	'Dies ist eine Zeile$N'
$P	neue Seite	16#0c	'Seitenende $P Seitenanfang'
$T	Tabulation	16#09	'Name$TGröße$TDatum'
$hh (*)	belieb. Zeichen	16#hh	'ABCD = $41$42$43$44'

(*) "hh" ist der Ascii-Code des Zeichens, als zweistelliger Hexadezimalausdruck.

B.2.3. Variablen

Variablen können Programm-**LOKAL** oder **GLOBAL** sein. Lokale Variablen können nur von einem einzigen Programm benutzt werden, globale Variablen dahingegen von allen Anwendungsprogrammen. Variablennamen müssen die folgenden Regeln beachten :
-Der Name darf nicht länger als **16** Zeichen sein,
- das erste Zeichen muß ein **Buchstabe** sein**,**
- darauffolgende Zeichen müssen **Buchstaben**, **Zahlen** oder '**_**' sein.

B.2.3.1. Reservierte Schlüsselwörter

Es folgt eine Liste reservierter Schlüsselwörter. Solche Bezeichner können nicht als Namen für Programme, Variablen, "C"-Funktionen oder -Funktionsbausteine benutzt werden:

A ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
B BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,
C CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
D DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,

E ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
F FALSE, FEDGE, FIND, FOR, FUNCTION,
G GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
I IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
J JMP, JMPC, JMPCN, JMPN, JMPNC,
L LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
M MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
N NE, NOT,
O OF, ON, OPERATE, OR, OR_MASK, ORN,
P PROGRAM
R R, REDGE, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR,
S S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM,
T TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP, TYPE,
U UDINT, UINT, ULINT, UNTIL, USINT,
V VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, ,VAR_OUTPUT,
W WHILE, WITH, WORD,
X XOR, XOR_MASK, XORN

Alle Schlüsselwörter, die mit einem Unterstrich-Zeichen ('_') beginnen, sind interne Schlüsselwörter und dürfen nicht in Textanweisungen benutzt werden.

B.2.3.2. Direkt dargestellte Variablen

ISaGRAF ermöglicht die Verwendung von **direkt dargestellten Variablen** in den Quellcodes der Programme, um einen freien Kanal darzustellen. Unter freien Kanälen versteht man Kanäle, welche nicht mit einer deklarierten E/A-Variablen verknüpft sind. Der Bezeichner einer direkt dargestellten Variablen beginnt stets mit dem Zeichen "**%**".
Es folgen die Benennungskonventionen einer direkt dargestellten Variablen für den Kanal einer einzelnen Karte. "***s***" ist die Steckplatznummer der Karte. "***c***" ist die Nummer des Kanals.

`%IXs.c` freier Kanal einer booleschen Eingangskarte
`%IDs.c` freier Kanal einer ganzzahligen Eingangskarte
`%ISs.c` freier Kanal einer Zeichenketten-Eingangskarte
`%QXs.c` freier Kanal einer booleschen Ausgangskarte

`%QDs.c` freier Kanal einer ganzzahligen Ausgangskarte
`%QSs.c` freier Kanal einer Zeichenketten- Ausgangskarte

Es folgen die Benennungskonventionen einer direkt dargestellten Variablen für den Kanal einer komplexen Baugruppe. "***s***" ist die Steckplatznummer der Baugruppe. "***b***" ist der Index der einzelnen Karte innerhalb der Baugruppe. "***c***" ist die Nummer des Kanals.

`%IXs.b.c` freier Kanal einer booleschen Eingangskarte
`%IDs.b.c` freier Kanal einer ganzzahligen Eingangskarte
`%ISs.b.c` freier Kanal einer Zeichenketten-Eingangskarte
`%QXs.b.c` freier Kanal einer booleschen Ausgangskarte
`%QDs.b.c` freier Kanal einer ganzzahligen Ausgangskarte
`%QSs.b.c` freier Kanal einer Zeichenketten-Ausgangskarte

Hier zwei Beispiele:

`%QX1.6` 6ter Kanal der Karte #1 (boolescher Ausgang)
`%ID2.1.7` 7ter Kanal der Karte #1 in der Baugruppe #2 (ganzzahliger Eingang)

Eine direkt dargestellte Variable kann nicht den Datentyp "**real**" haben.

B.2.3.3. Boolesche Variablen

Boolesch ist gleichbedeutend mit **logisch**. Boolesche Variablen können ausschließlich die Werte **TRUE** oder **FALSE** annehmen und werden in booleschen Ausdrücken benutzt. Eine boolesche Variable kann eines der folgenden **Attribute** besitzen :

Intern: vom Programm aktualisierte Speichervariable
Konstant: nur-lesen Speichervariable mit einem Initialwert
Eingang: an einen Sensor angeschlossene Variable (vom System aktualisiert)
Ausgang: an einen Aktor angeschlossene Variable

Achtung: Bei der Deklaration von booleschen Variablen können die Werte 'true' und 'false' durch Zeichenketten ersetzt werden, die jedoch ausschließlich für die Programmeinstellung bestimmt sind. Diese Zeichenfolgen dürfen nicht in der Programmierung benutzt werden, da sie keine **Definitionen** von äquivalenten Wörtern sind.

B.2.3.4. Analoge Variablen

Analog ist gleichbedeutend mit **kontinuierlich**. Diese Variablen nehmen vorzeichenbehaftete, ganzzahlige oder reale (Fließkomma-) Werte an. Folgende Formate sind verfügbar :

Ganzzahlig: vorzeichenbehaftete Ganzzahl auf 32 Bits : von **-2147483647** bis **+2147483647**
Real: Fließkommawert auf 32 Bits - Standard IEEE

(einfache Präzision)
1 Vorzeichenbit + 23 Mantisse-Bits + 8 Exponenten-Bits

Der Exponent einer REALEN, analogen Variablen muß zwischen einschließlich **-37** und **+37** liegen.
Analoge Variablen können eins der folgenden **Attribute** besitzen :

Intern:	vom Programm aktualisierte Speichervariable
Konstant:	nur-lesen Speichervariable mit einem Initialwert
Eingang:	an einen Sensor angeschlossene Variable (vom System aktualisiert)
Ausgang:	an einen Aktor angeschlossene Variable

Anmerkung: Wenn eine reale Variable an eine E/A-Vorrichtung angeschlossen ist, verarbeitet der entsprechende E/A-Pilot den äquivalenten, ganzzahligen Wert.
Achtung: Ganzzahlige und reale Variablen dürfen nicht in demselben analogen Ausdruck gemischt vorkommen.

B.2.3.5. Timer-Variablen

Ein Timer ist ein . Timer-Variablen nehmen zeitbezogene Werte an, die zwischen **0s** und **23h59m59s99** liegen und niemals negativ sein dürfen. Sie werden auf 32 Bits gespeichert. Ihre interne Darstellung ist eine Ganzzahl in Millisekunden.
Timer-Variablen können eins der folgenden **Attribute** besitzen:

Intern	vom Programm verwaltete Speichervariable, die vom ISaGRAF System aktualisiert wird
Konstant:	nur-lesen Speichervariable mit einem Initialwert

Achtung: Timer-Variablen können keine EINGANGS- oder AUSGANGS-Attribute besitzen.

Timer-Variablen können vom ISaGRAF System automatisch aktualisiert werden. Wenn ein Timer **aktiv** ist, wird sein Wert automatisch nach der Systemuhr inkrementiert. Die folgenden Anweisungen können benutzt werden, um einen Timer zu steuern :

TSTART:	startet die automatische Aktualisierung eines Timers
TSTOP:	stoppt die automatische Aktualisierung eines Timers

B.2.3.6. Zeichenkette-Variablen

Bei diesen Variablen handelt es sich um Variablen, die Zeichenketten enthalten. Die Länge der Kette hängt von den ausgeführten Operationen ab und beschränkt sich auf die maximale Länge, die bei ihrer Deklaration bestimmt wurde (maximal 255 Zeichen). Eine Zeichenketten-Variable kann die folgenden **Attribute** besitzen :

Intern:	vom Programm aktualisierte Speichervariable
Konstant:	nur-lesen Speichervariable mit einem Initialwert
Eingang:	an einen Sensor angeschlossene Variable (vom System aktualisiert)
Ausgang:	an einen Aktor angeschlossene Variable

Zeichenketten-Variablen können sämtliche Zeichen der Ascii-Tabelle enthalten (Codes von **0** bis **255**). Das Null-Zeichen (Code 0) ist innerhalb der Kette erlaubt. Bestimmte "C"-Funktionen der ISaGRAF Standard-Bibliothek können Zeichenketten, die Null-Zeichen enthalten, nicht richtig verarbeiten.

B.2.4. Kommentare

Im Quellcode der **ST**- und **AWL-**Textprogramme dürfen Kommentare frei eingefügt werden. Ein Kommentar beginnt mit dem Zeichen "**(***" und ended mit dem Zeichen "***)**". Kommentare können in einem **ST**-Programm an beliebigen Stellen eingesetzt werden und können sich über mehrere Zeilen erstrecken. Es folgen einige Beispiele :

```
counter := ivalue;   (* Zuweisung Hauptzähler *)
(* hier ein Kommentar
auf zwei Zeilen *)
c := counter (* Hauptzähler *) + base_value + 1;
```

Kommentare dürfen nicht verschachtelt werden. Die Zeichen "**(***" sind innerhalb der Kommentare nicht zugelassen.

Achtung: In der AWL-Sprache wird ein Kommentar nur dann akzeptiert, wenn er als letzter Komponent einer Anweisungszeile auftritt.

B.2.5. Definitionen von äquivalenten Wörtern

Im ISaGRAF System können konstante Ausdrücke, äquivalente Ausdrücke zu true oder false, Schlüsselwörter oder komplexe **ST**-Ausdrücke umdefiniert werden. Zu diesem Zweck wird einem Ausdruck ein entsprechender **Bezeichner** zugewiesen. Zum Beispiel :

JA	bedeutet	**TRUE**
PI	bedeutet	**3.14159**
OK	bedeutet	**(auto_modus AND NOT (alarm))**

Wenn eine Äquivalenz definiert ist, kann ihr **Bezeichner** im Quellcode eines Programms benutzt werden, um den entsprechenden Ausdruck zu ersetzen. Im folgenden ST-Programm werden die oben definierten Äquivalenzen verwendet :

```
If OK Then
  angle := PI / 2.0;
  isdone := JA;
End_if;
```

Definitionen können programm- **LOKAL**, **GLOBAL**, oder **ALLGEMEIN** sein. Lokalen Definitionen können nur von einem Programm benutzt werden. Globalen Definitionen können von allen Programmen eines Projekts benutzt werden. Allgemeine Definitionen können von allen Programmen aller Projekte benutzt werden. Beachten Sie, daß allgemeine Definitionen mit dem Archivmanager separat gespeichert werden können.

Warnung Wenn der gleiche Bezeichner zweimal mit unterschiedlichen **ST**-Äquivalenzen definiert wird, wird der zuletzt definierte Ausdruck benutzt. Zum Beispiel:

Definition:	**OPEN**	äquivalent zu	**FALSE**
	OPEN	äquivalent zu	**TRUE**
bedeutet:	**OPEN**	äquivalent zu	**TRUE**

Bei der Benennung der Definitionen müssen die folgenden Regeln beachtet werden:
- Der Name darf nicht länger als **16** Zeichen sein,
- das erste Zeichen muß ein **Buchstabe** sein,
- darauffolgende Zeichen müssen **Buchstaben**, **Zahlen**, oder '**_**' sein.

Warnung Eine Definition kann keine andere Definition in ihrem Ausdruck benutzen. Das folgende Beispiel ist ungültig:

PI is **3.14159**
PI2 is **PI*2**

Schreiben Sie die vollständige Äquivalenz unter Verwendung von Konstanten oder Variablen und Operationen:

PI is **6.28318**

B.3. AS-Sprache

Die **AS**-Sprache (Ablaufsprache) ist eine **graphische** Sprache, die für die Beschreibung von **sequentiellen** Operationen benutzt wird. Ein Prozeß besteht aus einer Reihe von bekannten **Schritten** (stabilen Zuständen), die durch **Transitionen** miteinander verbunden sind. Jeder Transition wird eine **boolesche Bedingung** zugeordnet. Die **Aktionen** in den Schritten werden in den **ST--**, **AWL-**, **KOP-** oder **FBS-**Sprachen beschrieben.

B.3.1. Format des AS-Netzwerks

Ein AS-Programm ist ein graphisches Netzwerk, bestehend aus **Schritten** und **Transitionen**, die durch **orientierte Verbindungslinien** verbunden werden. Mehrfach- Verbindungen werden durch Verzweigungen und Zusammenführungen dargestellt. Bestimmte Teile des Netzwerks können isoliert und im Hauptnetzwerk durch ein einziges Symbol, einen sogenannten **Makro-Schritt,** dargestellt werden. Die hauptsächlichen **graphischen Regeln** der AS-Sprache sind :

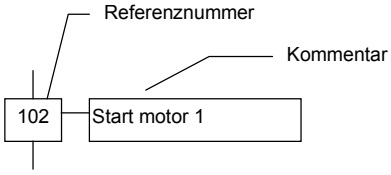
- Ein Schritt darf keinem anderen Schritt folgen.
- Eine Transition darf keiner anderen Transition folgen.

B.3.2. Grundkomponenten des AS-Netzwerks

Ein AS-Netzwerk besteht aus den folgenden Grundkomponenten (graphischen Symbolen) : Schritte und Initialisierungsschritte, Transitionen, orientierte Verbindungslinien und Sprungbefehle zu einem Schritt.

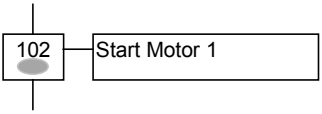
B.3.2.1. Schritte und Initialisierungsschritte

Ein Schritt wird durch ein **Quadrat** symbolisiert. Jeder Schritt erhält eine **Referenz**-Nummer, die in den Rahmen des Quadrats geschrieben wird. Eine allgemeine Beschreibung des Schritts wird in ein mit dem Schrittsymbol verbundenes Rechteck geschrieben. Diese Beschreibung ist ein **freier Kommentar** (keine Programmierungs-Sprache). Diese Informationen bilden die **Ebene 1** des Schritts :

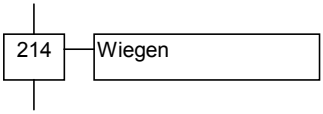


Bei der Ausführung zeigt eine **Zugriffsberechtigung** an, ob der Schritt **aktiv** ist :

Aktiver Schritt:

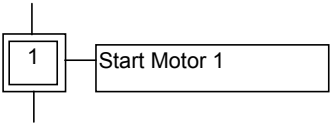


Inaktiver Schritt:



Die **initiale Situation** eines AS-Programms wird durch seine **Initialisierungsschritte** dargestellt. Initialisierungsschritte werden durch ein Quadrat mit **doppeltem Rahmen** symbolisiert. Beim Programm-Start wird jeder Initialisierungsschritt automatisch mit einer Zugriffsberechtigung versehen.

Initialisierungsschritt:



Jedes AS-Programm hat **mindestens einen** Initialisierungsschritt.

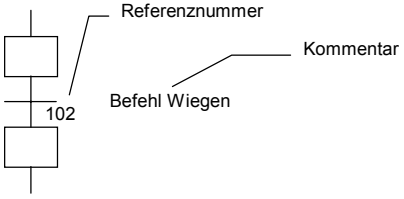
Schritte besitzen die folgenden Attribute (Diese Felder können von allen Sprachen aus getestet werden) :

GSnnn.x..... Schrittaktivität (boolescher Wert)
GSnnn.t..... Dauer der Schrittaktivität (Zeitwert)

(**nnn** ist die Referenznummer des Schritts)

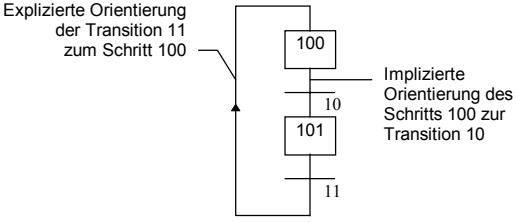
B.3.2.2. Transitionen

Eine Transition wird durch einen horizontalen Balken dargestellt, der eine Verbindungslinie durchkreuzt. Jede Transition besitzt eine **Referenz**-Nummer, die neben ihr Symbol geschrieben wird. Eine allgemeine Beschreibung der Transition wird rechts daneben eingegeben. Dies ist ein **freier Kommentar** (keine Programmierungs-Sprache). Diese Informationen bilden die **Ebene 1** der Transition :



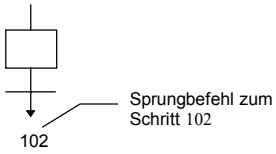
B.3.2.3. Orientierte Verbindungen

Orientierte Verbindungen werden durch einfache Linien zwischen den Schritten und Transitionen dargestellt. Wenn die Richtung (Orientierung) dieser Linien nicht ausdrücklich festgelegt wird, erfolgt die Verbindung von oben nach unten.

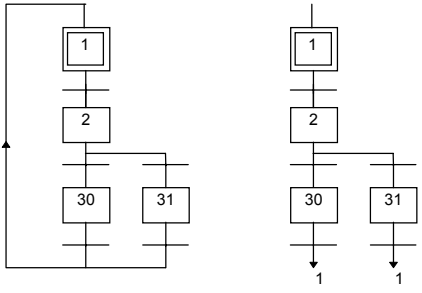


B.3.2.4. SprungbefehL zu einem Schritt

Ein Sprungbefehl-Symbol kann dazu benutzt werden, eine Transition mit einem Schritt zu verbinden, ohne daß eine Linie gezogen werden muß. Das Sprungbefehl-Symbol wird mit der Referenznummer des Zielschritts gekennzeichnet :



Sprungbefehl-Symbole dürfen nicht für Sprünge zu Transitionen verwendet werden. Es folgt das Beispiel eines Sprungbefehls - die Netzwerke sind äquivalent :



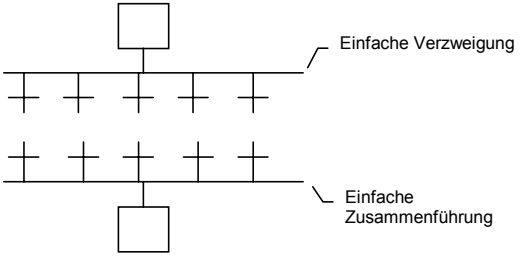
B.3.3. Verzweigungen und Zusammenführungen

Verzweigungen sind Mehrfach-Verbindungen von einzelnen AS-Symbolen aus (Schritten oder Transitionen) zu mehreren, anderen AS-Symbolen hin. Zusammenführungen sind Mehrfach-

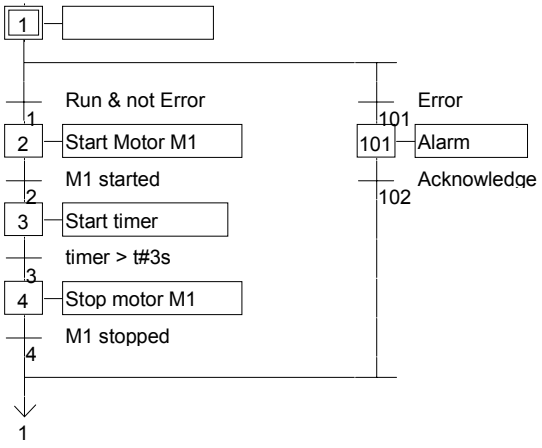
Verbindungen von mehreren AS-Symbolen her zu einem einzigen Symbol. Man unterscheidet zwischen zwei Arten von Verzweigungen und Zusammenführungen : einfache oder doppelte.

B.3.3.1. Einfache Verzweigungen

Eine einfache Verzweigung ist eine Mehrfach-Verbindung von einem Schritt aus zu mehreren Transitionen hin. Sie stellt mehrere Möglichkeiten in der Prozeßsequenz dar. Eine einfache Zusammenführung ist eine Mehrfach-Verbindung von mehreren Transitionen aus zu einem Schritt hin. Einfache Zusammenführungen werden im allgemeinen benutzt, um die offenen Zweige einer einfachen Verzweigung zu schließen. Einfache Verzweigungen und Zusammenführungen werden durch einfache horizontale Linien dargestellt.

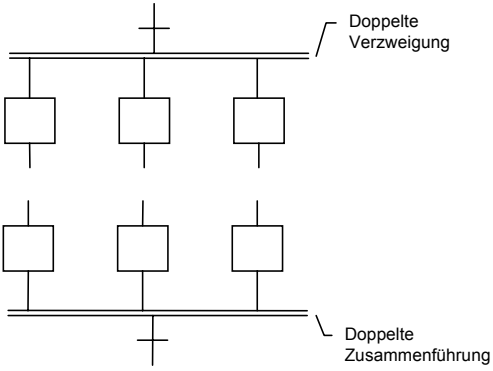


Achtung: Die Bedingungen, die mit den verschiedenen Transitionen am Anfang einer Verzweigung verbunden werden, **sind nicht impliziert exklusiv**. Die Exklusivität muß in jeder dieser Bedingungen expliziert detailliert werden, um sicher zu gehen, daß nur eine einzige Zugriffsberechtigung in einem der Zweige der Verzweigung erstellt wird, wenn dieser Zweig überschritten wird. Hier ein Beispiel mit einfachen Verzweigungen und Zusammenführungen :

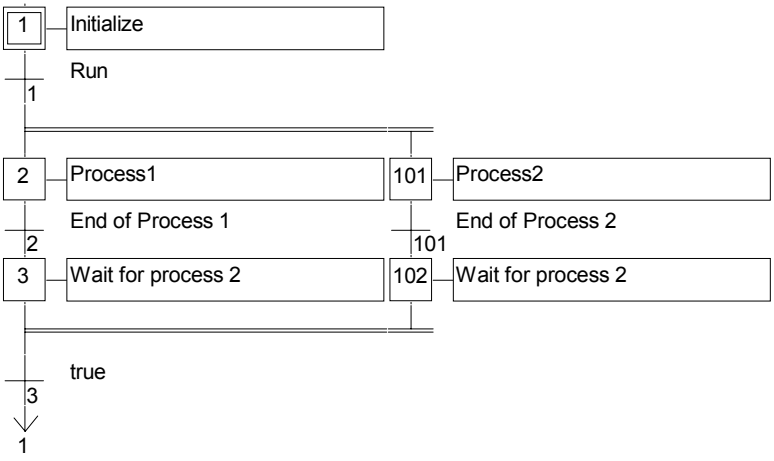


B.3.3.2. Doppelte Verzweigungen

Eine doppelte Verzweigung ist eine Mehrfach-Verbindung von einer Transition aus zu mehreren Schritten hin. Sie stellt im allgemeinen parallele Operationen in der Prozeßsequenz dar. Eine doppelte Zusammenführung ist eine Mehrfach-Verbindung von mehreren Schritten aus zu einer Transition hin. Eine doppelte Zusammenführung wird im allgemeinen benutzt, um offene Zweige einer doppelten Verzweigung zusammenzuführen. Doppelte Verzweigungen und Zusammenführungen werden durch doppelte, horizontale Linien dargestellt.



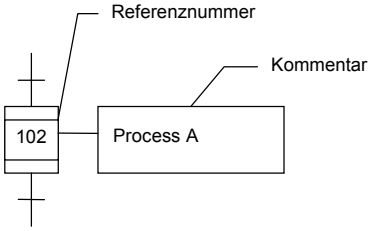
Hier ein Beispiel mit doppelten Verzweigungen und Zusammenführungen :



B.3.4. Makro-Schritte

Ein Makro-Schritt ist die einmalige Darstellung einer Gruppe von Schritten und Transitionen durch ein einmaliges Symbol. Dieser Makro-Schritt wird an einer anderen Stelle in demselben

AS-Programm beschrieben. Er erscheint als ein einmaliges Symbol im Hauptnetzwerk. Das folgende Symbol wird für die Darstellung eines Makro-Schritts benutzt :

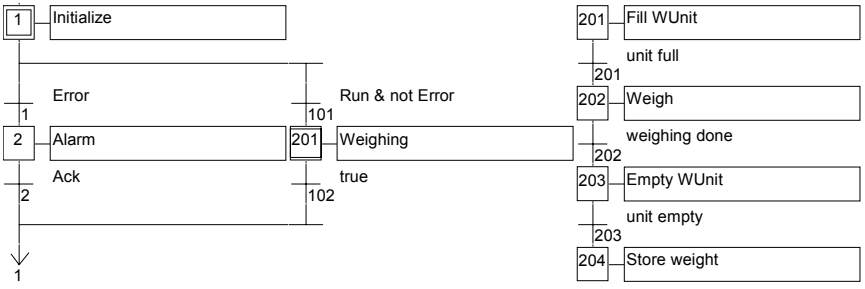


Die Referenznummer, die im Symbol des Makro-Schritts erscheint, ist die Referenznummer des ersten Schritts im Rumpf des Makro-Schritts. Der Rumpf eines Makro-Schritts beginnt mit einem **Anfangsschritt** und ended mit einem **Endschritt**. Das Netzwerk muß zusammenhängend sein. Ein Anfangsschritt darf keine vorherigen Verbindungen (keine vorangehenden Transitionen), ein Endschritt keine weiteren Verbindungen (keine folgenden Transitionen) haben. Das Symbol eines Makro-Schritts kann im Rumpf eines anderen Makro-Schritts erscheinen.

Achtung: Da ein Makro-Schritt eine **einmalige** Gruppe von Schritten und Transitionen darstellt, darf ein bestimmter Makro-Schritt nicht mehrmals in einem AS-Programm auftreten.

Hier das Beispiel eines Makro-Schritts :

```
(* AS-Programm, das einen Makro-Schritt einschließt *)

(* Hauptnetzwerk *)(* Rumpf des Makro-Schritts *)
```



B.3.5. Aktionen in den Schritten

Die **Ebene 2** eines AS-Schritts besteht aus einer ausführlichen Beschreibung der **Aktionen,** die ausgeführt werden, **wenn der Schritt aktiv ist**. Diese Beschreibung erfolgt unter Anwendung der **AS-Textstruktur** und anderen Sprachen, wie die **ST**-Sprache (Strukturierter Text). Hier die einzelnen Aktionstypen :

- Boolesche Aktionen
- Impuls-Aktionen (in ST programmiert)
- Nicht-gespeicherte Aktionen (in ST programmiert)
- AS-Aktionen

In einem Schritt können mehrere Aktionen beschrieben werden (eines oder mehrerer Typen). Die folgenden Strukturen ermöglichen die Integration von Verarbeitungen, die in anderen Sprachen geschrieben wurden :

- Aufruf eines Unterprogramms
- Einsatz des AWL-Codes (Anweisungsliste)

B.3.5.1. Boolesche Aktionen

Eine boolesche Aktion besteht daraus, einer booleschen Variablen das Aktivitätssignal eines Schritts (Schrittflagge) zuzuweisen. Die boolesche Variable muß eine interne oder Ausgangsvariable sein. Sie wird jedesmal forciert, wenn das Aktivitätssignal des Schritts seinen Zustand ändert.

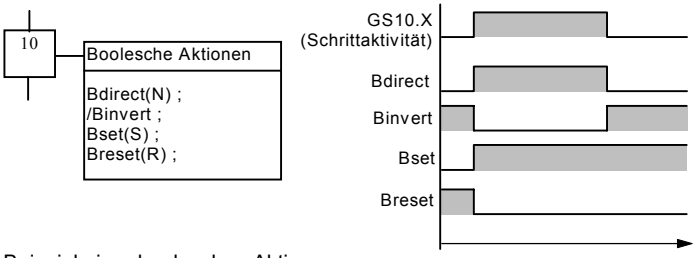
Hier die Syntax einer nicht-gespeicherten booleschen Aktion :

<boolesche_variable> (N);	Kopiert das Schritt-Aktivitätssignal in der Variablen
<boolesche_variable> ;	gleicher Effekt wie die vorherige Syntax (das Attribut N ist optionell)
/ <boolesche_variable> ;	kopiert die logische Inversion des Schritt-Aktivitätssignals in der Variablen

Andere Aktionen forcieren die Variable (zu false oder true), wenn der Schritt aktiv wird. Hier die Syntax dieser Aktionen :

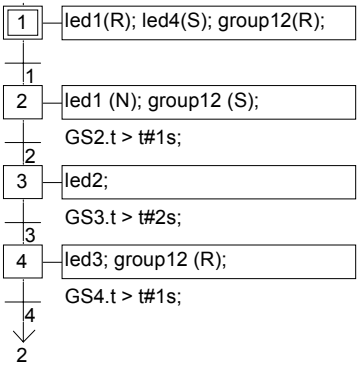
<boolesche_variable> (S) ;	forciert die Variable zu TRUE, wenn das Schritt-Aktivitätssignal den Zustand TRUE annimmt
<boolesche_variable> (R) ;	forciert die Variable zu FALSE, wenn das Schritt-Aktivitätssignal den Zustand TRUE annimmt

Die boolesche Variable muß eine INTERNE oder AUSGANGsvariable sein.
Die folgende AS-Programmierung bedeutet das im Chronogramm dargestellte Verhalten:



Beispiel einer booleschen Aktion :

(* AS-Programm mit booleschen Aktionen *)

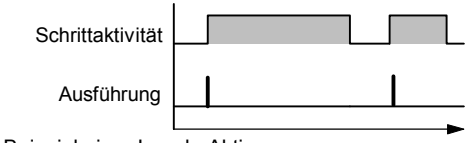


B.3.5.2. Impuls-Aktionen

Eine Impuls-Aktion ist eine ST- oder AWL-Anweisungsliste, die **ein einziges Mal** ausgeführt wird, wenn der Schritt **aktiv wird**. Die Anweisungen werden entsprechend der folgenden AS-Syntax geschrieben:

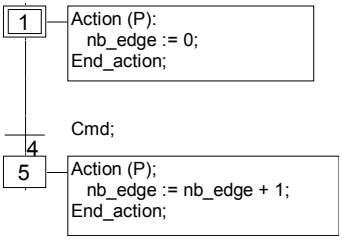
ACTION (P) :
(* ST-Anweisungen *)
END_ACTION ;

Hier das Steuerungsdiagramm einer Aktion vom Typ **P** :



Beispiel einer Impuls-Aktion :

(* AS-Programm mit Impuls-Aktionen *)

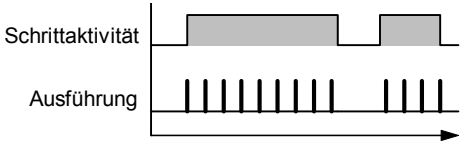


B.3.5.3. Nicht-gespeicherte Aktionen

Eine nicht-gespeicherte Aktion ist eine ST- oder AWL-Anweisungsliste, die bei **jedem Zyklus** ausgeführt wird, und zwar während der gesamten **Aktivitäts**-Dauer des Schritts. Die Anweisungen werden entsprechend der folgenden AS-Syntax geschrieben :

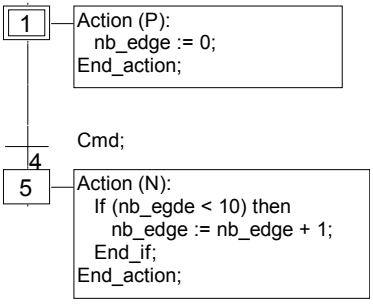
ACTION (N) :
(* ST-Anweisungen *)
END_ACTION ;

Hier das Steuerungsdiagramm einer Aktion vom Typ **N** :



Beispiel einer nicht-gespeicherten Aktion :

(* AS-Programm mit nicht-gespeicherten Aktionen *)



B.3.5.4. AS-Aktionen

Eine AS-Aktion ist eine AS-Tochtersequenz, die entsprechend der Entwicklung des Schrittaktivitätssignals gestartet oder gestoppt wird. Eine AS-Aktion kann mit den

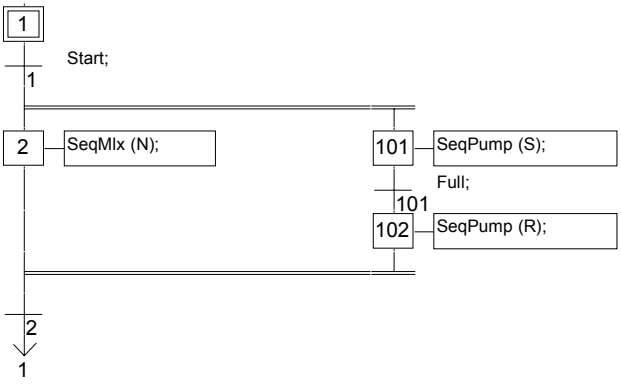
Aktionsqualifizierern **N** (Nicht-gespeichert), **S** (Set), oder **R** (Reset) geschrieben werden. Hier die Syntax der AS-Aktionen :

<sohn_programm> (N) ;	startet die Tochtersequenz, wenn der Schritt aktiv wird und stoppt die Tochersequenz, wenn der Schritt wird inaktiv wird
<sohn_programm> ;	gleicher Effekt, wie die vorherige Syntax (das Attribut N ist optionell)
<sohn_programm> (S) ;	startet die Tochtersequenz, wenn der Schritt aktiv wird nichts geschieht, wenn der Schritt wieder inaktiv wird
<sohn_programm> (R) ;	stoppt die Tochtersequenz, wenn der Schritt aktiv wird nichts geschieht, wenn der Schritt wieder inaktiv wird

Die in der Aktion spezifizierte AS-Sequenz muß ein **AS-Sohnprogramm** des in Bearbeitung befindlichen Programms sein. Man sollte nicht vergessen, daß die Benutzung der Qualifizierer **S** (Set) oder **R** (Reset) für eine AS-Aktion das gleiche bewirkt, wie die Anweisungen **GSTART** und **GKILL**, die in einer Aktion vom Typ P (Impuls-Aktion) in der **ST**-Sprache programmiert werden.

In dem folgenden Beispiel einer AS-Aktion wird das AS-Hauptprogramm "Princ" genannt. Es gibt zwei AS-Sohnprogramme, die SeqMlx und SeqPomp heißen. Hier die AS-Programmierung des Hauptprogramms :

(* AS-Programm mit AS-Aktionen *)



B.3.5.5. Aufruf von Funktionen und Funktionsbausteinen durch eine Aktion

Unterprogramme, Funktionen und Funktionsbausteine (in ST, AWL, KOP oder FBS geschrieben) oder "C"-Funktionen und "C"-Funktionsbausteine können direkt von einem AS-Aktionsblock aus aufgerufen werden, und zwar entsprechend der folgenden Syntax:

Für Unterprogramme, Funktionen und "C"-Funktionen:

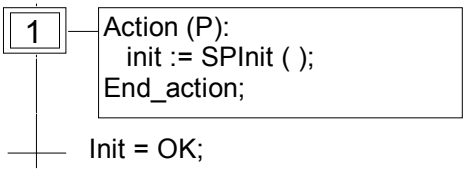
```
ACTION (P) :
     <ergebnis> := <unter_programm> ( ) ;
END_ACTION;

oder

ACTION (N) :
     <ergebnis> := <unter_programm> ( ) ;
END_ACTION;
```

Für Funktionsbausteine in "C" oder in ST, AWL, KOP, FBS:

```
      ACTION (P) :
            Fbinst(in1, in2);
            ergebnis1 := Fbinst.out1;
            ergebnis2 := Fbinst.out2;
      END_ACTION;

oder

      ACTION (N) :
            Fbinst(in1, in2);
            ergebnis1 := Fbinst.out1;
            ergebnis2 := Fbinst.out2;
      END_ACTION;
```

Die Syntax wird im Abschnitt "ST-Sprache" ausführlich beschrieben.
Es folgt das Beispiel eines Unterprogramm-Aufrufs in einem Aktionsblock :

(* AS-Programm mit Aufruf eines Unterprogramms in einer Aktion *)



B.3.5.6. Schreibkonventionen der AWL-Sprache

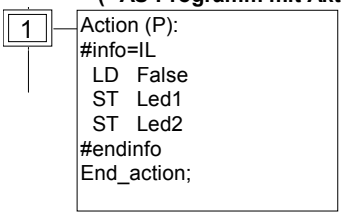
Die Programmierung in der **AWL**-Sprache (Anweisungsliste) kann direkt in einen Aktionsblock eingesetzt werden, und zwar entsprechend der folgenden Syntax :

```
ACTION (P) :   (* oder N *)
#info=IL
     <Anweisung>
     <Anweisung>
     ....
#endinfo
```

```
END_ACTION;
```

Die Schlüsselwörter "**#info=IL**" und "**#endinfo**" müssen exakt in dieser Form eingegeben werden (Groß- und Kleinschreibung beachten). Fügen Sie keine Leerzeichen oder Tabs vor oder nach diesen Schlüsselwörtern ein. Hier das Beispiel einer in der AWL-Sprache programmierten Aktion :

(* AS-Programm mit Aktion, programmiert in AWL *)



B.3.6. Mit Transitionen verbundene Bedingungen

Mit jeder Transition kann ein **boolescher Ausdruck** verbunden werden, der das Überschreiten der Transition bedingt. Diese Bedingungen, die die **Ebene 2** der Transition ausmachen, werden im allgemeinen in der ST-Sprache geschrieben. Andere Schreibkonventionen sind zugelassen :

- Schreibkonvention in ST
- Schreibkonvention in KOP
- Schreibkonvention in AWL
- Aufruf von Unterprogrammen

Achtung: Wenn einer Transition keine Bedingung zugeordnet wird, ist ihre vorgegebene Rezeptivität **TRUE**.

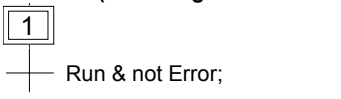
B.3.6.1. Schreibkonventionen der ST-Sprache

Die **ST**-Sprache (Strukturierter Text) kann verwendet werden, um Transitionen zugeordnete **Bedingungen** zu beschreiben. Die Ausdrücke müssen vom Typ **boolesch** sein und mit einem **Semikolon** enden, entsprechend der folgenden Syntax :

< boolescher_ausdruck > ;

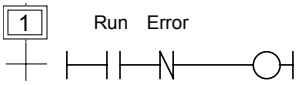
Ein Ausdruck kann ein konstanter Wert TRUE oder FALSE, eine boolesche interne oder Eingangsvariable oder eine Kombination von Variablen sein, die eine boolesche Größe darstellen. Hier das Beispiel einer in der ST-Sprache programmierten Transition :

(* AS-Programm mit Bedingungen, programmiert in ST *)



B.3.6.2. Schreibkonventionen der KOP-Sprache

Die **Kontaktplan**-Sprache (KOP) kann benutzt werden, um die mit einer Transition verbundenen **Bedingung** zu beschreiben. Das Diagramm besteht aus einem Rung mit einer Spule. Der Wert der Spule stellt den Wert der Transition dar.
Es folgt ein Beispiel einer KOP-Programmierung für Transitionen:



B.3.6.3. Schreibkonventionen der AWL-Sprache

Die **AWL**-Sprache (Anweisungsliste) kann unmittelbar für die Programmierung von Transitionsbedingungen verwendet werden, und zwar entsprechend der folgenden Syntax :

#info=IL
<Anweisung>
<Anweisung>
....
#endinfo

Der Wert, der am Ende der programmierten Sequenz im **aktuellen Ergebnis** (AWL-Verzeichnis) erscheint, ist das Ergebnis der mit der Transition verbundenen Bedingung :

Aktuelles Ergebnis = 0 ➔ Bedingung = **FALSE**
Aktuelles Ergebnis <> 0 ➔ Bedingung = **TRUE**

Die Schlüsselwörter "**#info=IL**" und "**#endinfo**" müssen exakt in dieser Form eingegeben werden (Groß- und Kleinschreibung beachten). Setzen Sie keine Leerzeichen oder Tabs vor oder nach diesen Schlüsselwörtern ein. Hier das Beispiel einer in der AWL-Sprache programmierten Bedingung :

(* AS-Programm mit Transitionen, programmiert in AWL *)


```
#info=IL
  LD  Run
  &N  Error
#endinfo
```

B.3.6.4. Aufruf eines Unterprogramms durch eine Transition

Unterprogramme (in FBS, KOP, ST, oder AWL geschrieben) können aufgerufen werden, um die mit einer Transition verbundene Bedingung zu bewerten, und zwar entsprechend der folgenden Syntax :

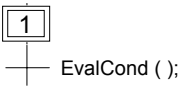
< unter_programm > ( ) ;

Der vom Unterprogramm zurückgegegebene Wert ist das Ergebnis der Überschreitungsbedingung der Transition :

zurückgegebener Wert = FALSE ➔ Bedingung = **FALSE**
zurückgegebener Wert = TRUE ➔ Bedingung = **TRUE**

Es folgt das Beispiel eines Unterprogramm-Aufrufs in einer Transition :

(* AS-Programm mit Unterprogramm-Aufruf *)



B.3.7. AS-Entwicklungsregeln

Die folgenden **fünf** Entwicklungsregeln sind in der AS-Sprache definiert :

Initiale Situation
Die initiale Situation wird durch **Initialisierungsschritte** dargestellt, die den Prozeßzustand beim Anwendungstart definieren. Ein AS-Programm besitzt **mindestens einen** Initialisierungsschritt.

Überschreitung einer Transition
Eine Transition ist entweder **gültig** oder **ungültig**. Sie ist gültig, wenn alle unmittelbar vorangehenden Schritte **aktiv** sind.
Eine Transition kann nicht überschritten werden, wenn :
- sie gültig ist und
- die mit ihr verbundene Bedingung true ist.

Entwicklung von aktiven Schritten
Die Überschreitung einer Transition zieht gleichzeitig die Aktivierung aller unmittelbar darauffolgenden Schritte und die Deaktivierung aller unmittelbar vorangehenden Schritte nach sich.

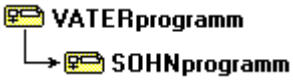
Gleichzeitige Überschreitung von Transitionen
Man kann doppelte Linien benutzen, um Transitionen darzustellen, die gleichzeitig überschritten werden sollen. Wenn diese Transitionen getrennt dargestellt werden, kann der Aktivitätstest der vorangehenden Schritte (GSnnn.x) in ihren Bedingungen eingeschlossen werden.

Gleichzeitige Aktivierung und Deaktivierung
Wenn ein Schritt in einem Sequenzablauf gleichzeitig aktiviert und deaktiviert werden soll, ist die Aktivierung prioritär.

B.3.8. Hierarchie der AS-Programme

Mit dem ISaGRAF System können AS-Programme in einer vertikalen Struktur geschrieben werden. Die AS-Programme werden in einem hierarchischen System angeordnet. Jedes Programm kann andere AS-Programme steuern (starten, stoppen...). Jedes AS-Programm

kann andere AS-Programme aufrufen, die sogenannten **Sohn**-Programme. AS-Programme sind in einer **hierarchischen Baumstruktur** miteinander verknüpft und stehen in einer "**Vater-Sohn**" Beziehungen zueinander :



Für diese Hierarchie gelten die folgenden Grundregeln :

- AS-Programme ohne Vaterprogramme heißen "**Hauptprogramme**".
- Hauptprogramme werden beim Anwendungsstart aktiviert.
- Ein Vaterprogramm kann mehrere Sohnprogramme besitzen.
- Ein Sohnprogramm darf nur ein einziges Vaterprogramm besitzen.
- Ein Sohnprogramm kann nur von seinem Vaterprogramm gesteuert werden.
- Ein Vaterprogramm kann nicht die Söhne seiner Sohnprogramme (seine Enkel) steuern.

AS-Vaterprogramme steuern ihre AS-Sohnprogramme mit den folgenden Aktionen :

Starten (GSTART) Startet das Sohnprogramm : aktiviert jeden seiner Initialisierungsschritte. Die Sohnprogramme des gestarteten Programms werden nicht automatisch mit ihm gestartet.
Stoppen (GKILL) Stoppt das Sohnprogramm, indem alle aktiven Schritte deaktiviert werden. Alle seine Sohnprogramme werden ebenfalls gestoppt.
Anhalten (GFREEZE) Deaktiviert alle aktiven Schritte des Sohnprogramms und speichert deren Position, um sie erneut starten zu können.
Erneut starten (GRST) Startet ein AS-Sohnprogramm erneut, wobei alle Schritte wieder aktiviert werden, die beim Anhalten des Programms deaktiviert wurden.
Testen (GSTATUS) Liefert den Zustand (angehalten, aktiv, inaktiv) des Programms.

B.4. Flußdiagramm-Sprache

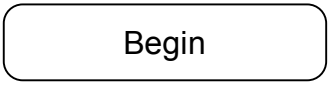
Das **Flußdiagramm** (**FD**) ist eine grafische Sprache für die Beschreibung **sequentieller Operationen**. Flußdiagramme bestehen aus **Aktionen** und **Tests**. Der Datenfluß zwischen den Aktionen und Tests wird durch **gerichtete Verbindungslinien** dargestellt. Die Darstellung von Verzweigungen und Zusammenführungen erfolgt durch Mehrfachverbindungen. Die Aktionen und Tests können in den Programmiersprachen ST, KOP und AWL beschrieben werden, und Funktionen und Funktionsbausteine, die in beliebigen anderen Sprachen (außer AS) geschriebenen wurden, aufrufen. Flußdiagramme können andere Flußdiagramme aufrufen. Das aufgerufene FD-Programm ist ein **Unterprogramm** des aufrufenden FD-Programms.

B.4.1. FD-Komponenten

Es folgen die grafischen Komponenten der Flußdiagramm-Sprache:

Anfang des FD-Netzwerks

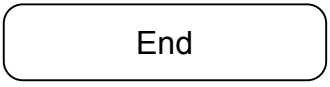
Am Anfang eines FD-Programms steht das "**Begin**"-Symbol. Dieses Symbol ist einmalig und darf nicht weggelassen werden. Es stellt bei seiner Aktivierung den Initialzustand des Flußdiagramms dar und sieht folgendermaßen aus:



Das "Begin"-Symbol muß an ein anderes Objekt des Flußdiagramms (nach unten hin) angeschlossen werden, sonst ist das Flußdiagramm ungültig.

Ende des FD-Netzwerks

Das "**End**"-Symbol bildet den Abschluß eines Flußdiagramms. Es ist einmalig und darf nicht weggelassen werden. Dieses Symbol stellt den abschließenden Zustand des Flußdiagramms dar und muß an sein Ende gesetzt werden, selbst wenn das Diagramm in einer unendlichen Schleife ausgeführt wird und kein eigentlicher Anschluß zum "End"-Symbol besteht. Das "End"-Symbol sieht so aus:



Normalerweise besitzt das "End"-Symbol einen Anschluß nach oben zu den anderen Objekten eines Flußdiagramms. Es gibt auch Flußdiagramme, bei denen das "End"-Symbol nicht angeschlossen ist (unendliche Schleifenausführung), in diesem Fall bleibt es jedoch am Ende des Diagramms sichtbar.

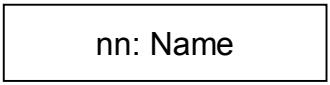
FD-Datenflußverbindungen

Die sogenannte **Datenflußverbindung** ist eine Linie, die den Datenfluß zwischen zwei Diagrammpunkten darstellt. Sie endet in einem Pfeil. Hier die Abbildung einer solchen Verbindung:

Es ist nicht möglich, zwei Verbindungen am gleichen Quellenpunkt anzuschließen.

FD-Aktionen

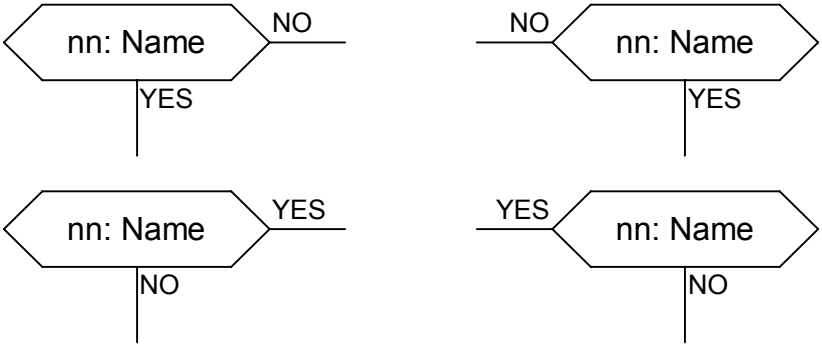
Die **Aktionssymbole** stellen auszuführende Aktionen dar. Aktionen werden jeweils mit einer Nummer und einem Namen gekennzeichnet. Es folgt die Abbildung eines "Aktionssymbols":



Zwei unterschiedliche Objekte eines Flußdiagramms können nicht den gleichen Namen oder die gleiche logische Nummer besitzen. Die Programmiersprache für Aktionen ist wahlweise ST, KOP oder AWL. Eine Aktion besitzt stets zwei Anschlußverbindungen, davon führt eine zur Aktion hin und eine andere von der Aktion weg.

FD-Tests

Ein **FD-Test** stellt eine boolesche **Bedingung** dar und wird mit einer Nummer und einem Namen gekennzeichnet. Entsprechend der Bewertung des zugeordneten ST-, KOP- oder AWL-Ausdrucks wird der Datenfluß zu einem "YES" oder "NO" Pfad geleitet. Es folgen die verschiedenen Möglichkeiten, solche Testsymbole zu zeichnen:



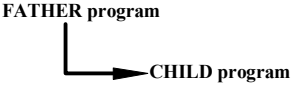
Zwei unterschiedliche Objekte eines Flußdiagramms können nicht den gleichen Namen oder die gleiche logische Nummer besitzen. Die Programmierung eines Tests ist wahlweise:
- ein Ausdruck in ST,
- ein einzelnes Rung im KOP, wobei kein Symbol mit der einzigen Spule verbunden ist, oder
- mehrere Anweisungen in AWL. Das AWL-Register (oder aktuelles Ergebnis) wird benutzt, um die Bedingung zu bewerten.

Beim Programmieren im ST-Text kann wahlweise ein Semikolon auf den Ausdruck folgen. Beim Programmieren im KOP stellt die einzige Spule den Wert der Bedingung dar. Eine Bedingung ist entweder:
- 0 oder FALSE (leitet den Datenfluß zu NO)
- 1 oder TRUE (leitet den Datenfluß zu YES)

Ein Test besitzt stets eine Eingangsverbindung und zwei definierte Ausgangsanschlüsse.

FD-Unterprogramme

Das System ermöglicht die Beschreibung einer Hierarchiestruktur. Die FD-Programme werden in einer **hierarchischen Baumstruktur** angeordnet. Jedes FD-Programm kann andere FD-Programme aufrufen. Diese Programme bezeichnet man als **Sohnprogramme** des aufrufenden FD-Programms. FD-Programme, die andere FD-Programme aufrufen, werden **Vaterprogramme** genannt. Die FD-Programme werden in einer hierarchischen Baumstruktur miteinander verknüpft und stehen in einer "Vater-Sohn"-Beziehung zueinander:



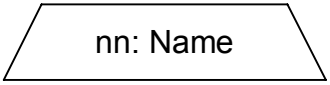
Das **Unterprogramm**-Symbol in einem Flußdiagramm stellt den Aufruf eines untergeordneten Flußdiagramms dar. Die Ausführung des aufrufenden FD-Programms wird unterbrochen, bis die Ausführung des Unterprogramms beendet ist. Ein FD-Unterprogramm wird mit einer Nummer und einem Namen gekennzeichnet, wie andere Programme, Funktionen oder Funktionsbausteine. Es folgt das Symbol eines "Unterprogrammaufrufs":

Zwei unterschiedliche Objekte eines Flußdiagramms können nicht die gleiche logische Nummer besitzen. Die Grundregeln der FD-Hierarchiestruktur sind:
- FD-Programme, die keinem Vaterprogramm untergeordnet sind, bezeichnet man als FD-Hauptprogramme.
- FD-Hauptprogramme werden beim Anwendungsstart vom System aktiviert.
- Ein Programm kann mehrere Sohnprogramme besitzen.
- Ein Sohnprogramm kann nicht mehrere Vaterprogramme besitzen.
- Ein Sohnprogramm kann nur von seinem Vaterprogramm aufgerufen werden.
- Ein Programm kann keine Sohnprogramme seiner eigenen Sohnprogramme aufrufen.

Ein und dasselbe Unterprogramm kann mehrmals im Vaterdiagramm erscheinen, dagegen kann es nicht in verschiedenen Zweigen einer parallelen Verzweigung auftreten. Der Aufruf eines Unterprogramms im Flußdiagramm stellt die vollständige Ausführung des Unterdiagramms dar. Solange das Sohndiagramm ausgeführt wird, bleibt die Ausführung des Vaterdiagramms unterbrochen. Für die Unterprogramm-Aufrufbausteine gelten die gleichen Anschlußregeln wie für die Aktionen.

Spezifische E/A-Aktionen im FD

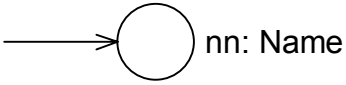
Das Symbol einer **spezifischen E/A-Aktion** stellt auszuführende Aktionen dar. Wie alle Aktionen werden auch die spezifischen E/A-Aktionen jeweils mit einer Nummer und einem Namen gekennzeichnet. Standardaktionen und spezifische E/A-Aktionen benutzen die gleiche Semantik. Die spezifischen E/A-Aktionen sollen das Flußdiagramm leserlicher machen und nicht-portierbare Abschnitte des Diagramms hervorheben. Die Verwendung von spezifischen E/A-Aktionen ist optionell. Das Symbol einer "spezifischen E/A-Aktion" sieht so aus:



Spezifische E/A-Aktionsbausteine haben die gleichen Eigenschaften wie Standardaktionen, sie werden mit denselben Programmiersprachen geschrieben und die Anschlußregeln sind identisch.

FD-Anschlüsse

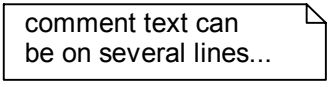
Ein **FD-Anschluß** stellt die Verbindung von zwei Diagrammpunkten her, ohne daß diese Verbindung gezeichnet werden muß. Ein solcher Anschluß wird als Kreis dargestellt und ist an die Quelle des Datenflusses angeschlossen. Je nach Richtung des Datenflusses wird das Anschlußsymbol an der entsprechenden Seite mit der Identifizierung des Zielpunkts vervollständigt (im allgemeinen der Name eines Zielsymbols). Es folgt die grafische Darstellungsweise eines Anschlusses:



Das Anschlußziel ist stets ein bereits definiertes FD-Symbol. Das Zielsymbol wird durch seine logische Nummer identifiziert.

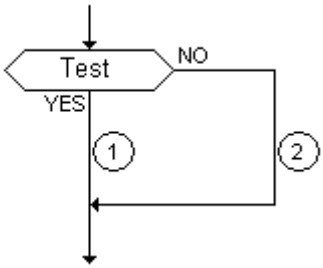
FD-Kommentare

Die **Kommentarfelder** enthalten Text, der keinen Einfluß auf die Semantik des Flußdiagramms nimmt, und können an beliebigen freien Stellen im Dokumentfenster eines Flußdiagramms eingefügt werden. Kommentare dienen der Programmdokumentation. Das "Kommentarsymbol" sieht folgendermaßen aus:



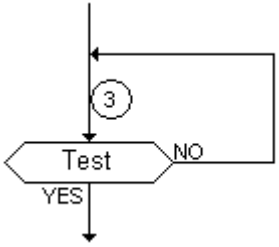
B.4.2. Komplexe FD-Strukturen

Dieser Abschnitt zeigt Beispiele **komplexer Strukturen**, wie sie in einem Flußdiagramm definiert werden können. Solche Strukturen sind Kombinationen von miteinander verbundenen Grundobjekten.



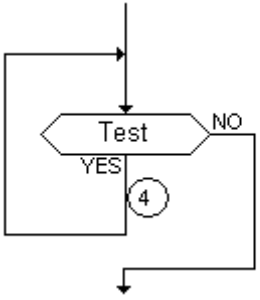
IF / THEN / ELSE

(1) hier können "THEN" Aktionen eingefügt werden
(2) hier können "ELSE" Aktionen eingefügt werden



REPEAT / UNTIL

(3) hier können wiederholte Aktionen eingefügt werden



WHILE / DO

(4) hier können wiederholte Aktionen eingefügt werden

B.4.3. Dynamisches Verhalten des FD

Die **Ausführung** eines Flußdiagramms kann folgendermaßen erklärt werden:

- Das "Begin"-Symbol belegt einen Zielsystemzyklus.
- Das "End"-Symbol belegt einen Zielsystemzyklus und beendet die Diagrammausführung. Wenn dieses Symbol erreicht ist, werden keine Aktionen des Flußdiagramms mehr ausgeführt.
- Die Ausführung einer Aktion belegt einen Zielsystemzyklus.

Hinweis: Im Gegensatz zu AS-Aktionen sind FD-Aktionen keine stabilen Zustände. Die Anweisungen werden nicht wiederholt, während das Aktionssymbol markiert ist.

B.4.4. FD prüfen

Außer der Syntaxregeln der zugeordneten ST-, KOP- und AWL-Programmierung gibt es noch einige andere **Syntaxregeln**, die sich auf das Flußdiagramm selbst beziehen. Folgende Regeln sind zu beachten:
- Alle "Anschlußpunkte" aller Symbole müssen angeschlossen werden. (Nur der Anschluß zum "End"-Symbol darf weggelassen werden.)
- Alle Symbole müssen miteinander verbunden werden. (Isolierte Abschnitte sind unzulässig.)
- Alle Anschlüsse müssen ein gültiges Ziel haben.

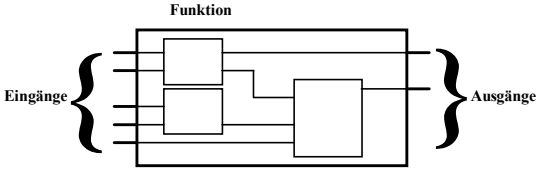
Andere Syntaxfehler können auftreten:
-Leere Aktionen (keine Programmierung) werden beim Runtime-Scheduling als Schritte betrachtet.
- Leere Tests (keine Programmierung) werden als "immer true" betrachtet.

B.5. FBS-Sprache

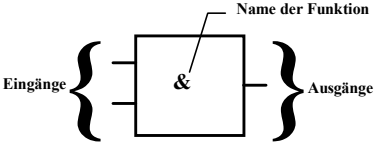
Die **FBS**-Sprache (Funktionsbaustein-Sprache) ist eine graphische Sprache, mit der man komplexe Gleichungen unter Anwendung der elementaren Funktionen der ISaGRAF Bibliotheken aufbauen kann.

B.5.1. Format des FBS-Diagramms

Ein FBS-Diagramm beschreibt eine Funktion zwischen **Eingangsvariablen** und **Ausgangsvariablen**. Diese Funktion wird wie ein Netzwerk aus **elementaren Funktionsbausteinen** aufgebaut. Die Ein- und Ausgangsvariablen werden mit den Funktionsbausteinen durch **Verbindungslinien** verbunden. Der Ausgang eines Funktionsbausteins kann an den Eingang eines anderen Funktionsbausteins angeschlossen werden.



Eine durch ein FBS-Diagramm dargestellte, komplette Funktion wird mit Hilfe der **elementaren** Funktionsbausteine der ISaGRAF Bibliothek konstruiert. Jeder elementare Funktionsbaustein besitzt eine vordefinierte Anzahl an **Anschlußpunkten** für seine Ein- und Ausgänge. Ein Funktionsbaustein wird durch ein **Rechteck** dargestellt. Seine Eingänge werden **links** am Rechteck, seine Ausgänge **rechts** am Rechteck angeschlossen. Ein elementarer Funktionsbaustein führt eine elementare **Funktion** zwischen seinen Ein- und Ausgängen aus. Der Name der ausgeführten Funktion erscheint im Rechteck des Bausteins. Die Ein- und Ausgänge des Bausteins sind jeweils von einem bestimmten **Typ**.



Die Eingangsvariablen eines FBS-Diagramms werden an die Eingänge der Funktionsbausteine angeschlossen. Der Typ einer jeden Variablen muß dem Typ des entsprechenden Funktionsbaustein-Eingangs entsprechen. Eingänge für ein FBS-Diagramm können **konstante** Ausdrücke oder beliebige **interne**, **Ein-** oder **Ausgangsvariablen** sein.

Die Ausgangsvariablen eines FBS-Diagramms werden an die Ausgänge der Funktionsbausteine angeschlossen. Der Typ einer jeden Variablen muß dem Typ des

entsprechenden Funktionsbaustein-Ausgangs Rechnung tragen. Ausgänge für ein FBS-Diagramm kann eine beliebige **interne** oder **Ausgangsvariable** oder der Name des bearbeiteten Programms sein (nur für **Unterprogramme**). Wenn der Name des bearbeiteten Programms als Diagramm-Ausgang benutzt wird, stellt er eine Zuweisung des vom Unterprogramm zurückgegebenen Werts dar.

Die Ein- und Ausgangsvariablen und die Ein- und Ausgänge eines Funktionsbausteins werden durch **Verbindungslinien** miteinander verbunden. Sie dienen der Verbindung von zwei Diagrammpunkten :

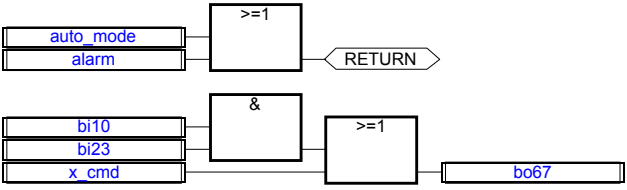
- Eingangsvariable mit einem Bausteineingang
- Bausteinausgang mit dem Eingang eines anderen Bausteins
- Bausteinausgang mit einer Ausgangsvariablen

Diese Verbindungen sind **orientiert** : Sie transportieren eine Information von ihrem linken zu ihrem rechten Ende. Die linken und rechten Enden einer Verbindungen müssen mit Elementen **des gleichen Typs** verbunden werden.

Mehrfach-Verbindungslinien nach rechts werden benutzt, um eine Information vom linken Ende einer Verbindung zu mehreren Punkten nach rechts zu transportieren. Alle Enden der Verbindungslinie müssen vom **gleichen Typ** sein.

B.5.1.1. Die 'RETURN'-Anweisung

Das Schlüsselwort "**<RETURN>**" kann als Diagramm-Ausgang erscheinen. Es muß an den booleschen Ausgang eines Funktionsbausteins angeschlossen werden. Die RETURN-Anweisung stellt eine **bedingte Beendung** der Programmausführung dar : Wenn der mit ihr verbundene Ausgang den booleschen Zustand **TRUE** annimmt, wird das Ende des Diagramms nicht interpretiert. Es folgt das Beispiel einer RETURN-Anweisung :



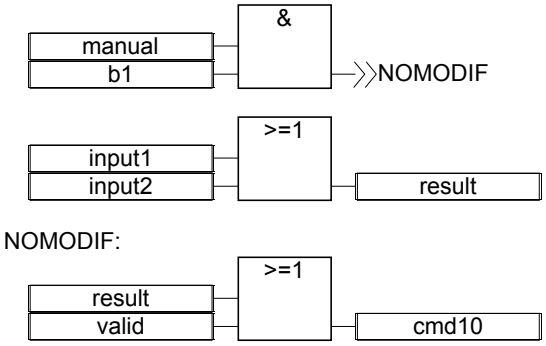

```
(* ST-Äquivalenz: *)
If auto_mode OR alarm Then
        Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;
```

B.5.1.2. Sprünge und Marken

Marken und bedingte Sprünge werden verwendet, um die Diagrammausführung zu steuern. Rechts vom Marken- oder Sprungsymbol kann keine Verbindung vorgenommen werden.

>>LAB Sprung zu einer Marke (der Name der Marke ist "LAB")
LAB: Definition einer Marke (genannt "LAB")

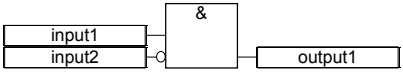
Wenn die Verbindung **links** vom Sprungsymbol den booleschen Zustand **TRUE** annimmt, wird die Programmausführung zur entsprechenden Marke umgeleitet. Das folgende Beispiel veranschaulicht die Anwendung der Sprünge und Marken:




```
(* AWL-Äquivalenz: *)
            ld      manual
            and     b1
            jmpc    NOMODIF
            ld      input1
            or      input2
            st      result
NOMODIF:    ld      result
            or      valid
            st      cmd10
```

B.5.1.3. Boolesche Inversion

Eine einfache Verbindungslinie, die an den booleschen Eingang eines Funktionsbausteins angeschlossen ist, kann in einer **booleschen Inversion** enden. Sie wird durch einen kleinen Kreis dargestellt. Wenn eine boolesche Inversion verwendet wird, müssen die linken und rechten Enden der entsprechenden Verbindunglinie vom Typ **boolesch** sein. Hier ein Beispiel:

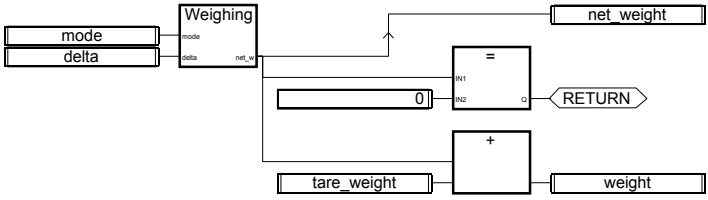



```
(* ST-Äquivalenz: *)
output1 := input1 AND NOT (input2);
```

B.5.1.4. Aufruf von Funktionen und Funktionsbausteinen von der FBS

Die FBS-Sprache ermöglicht den Aufruf von Unterprogrammen , Funktionen oder Funktionsbausteinen. Unterprogramme, Funktionen oder Funktionsbausteine werden als ein Funktionskasten dargestellt. Der Name im Kasten ist der Name des Unterprogramms, der Funktion oder des Funktionsbausteins.
Bei Unterprogrammen und Funktionen ist der Rückgabewert der einzige Ausgang des Funktionskastens.
Funktionsbausteine können mehrere Ausgänge haben.

(* Beispiel eines FBS-Programms, das einen Baustein vom Typ UNTERPROGRAMM benutzt *)

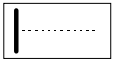


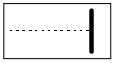

```
(* ST-Äquivalenz: *)
net_weight := Weighing (mode, delta); (* Unterprogramm-Aufruf *)
If (net_weight = 0) Then Return; End_if;
weight := net_weight + tare_weight;
```

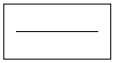
B.6. KOP-Sprache

Die **KOP**-Sprache (Kontaktplan) ist eine graphische Darstellung boolescher Gleichungen, in der **Kontakte** (als Eingänge) und **Spulen** (als Ausgänge) miteinander kombiniert werden.

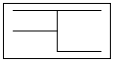
In der KOP-Sprache können **boolesche** Daten mit Hilfe von **graphischen Symbolen**, die in ein Diagramm eingesetzt werden, manipuliert werden. Die Symbole des KOP-Diagramms werden wie Elemente eines elektrischen Kontaktplans behandelt. KOP-Diagramme werden links und rechts durch **Stromschienen** begrenzt. Hier die graphischen Grundkomponenten eines KOP-Diagramms :

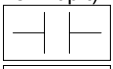

---------- Linke Stromschiene (mit horizontaler Verbindung)

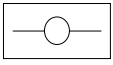

---------- Rechte Stromschiene (mit horizontaler Verbindung)


---------- Horizontale Verbindungslinie


---------- Vertikale Verbindungslinie

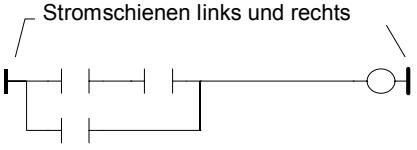

---------- Vertikale und horizontale Mehrfach-Verbindungen (alle verknüpft)


---------- Mit einer Variablen verbundener Kontakt

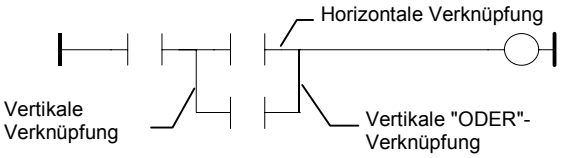

---------- Mit einem Ausgang oder einer internen Variablen verbundene Spule

B.6.1. Stromschienen

KOP-Diagramme werden links und rechts durch vertikale Linien, die sogenannten **linken und rechten Stromschienen,** begrenzt.



Die einzelnen Symbole eines KOP-Diagramms werden miteinander, sowie mit den Stromschienen durch **Verbindungslinien** verbunden. Verbindungen erfolgen entweder vertikal oder horizontal.



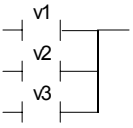
Jedes Verbindungssegment kann die booleschen Zustände **FALSE** oder **TRUE** annehmen. Der Zustand einer Verbindung propagiert sich in seine sämtlichen, rechten Enden. Horizontale Verbindungen, die an die **linke Stromschiene** angeschlossen sind, haben den booleschen Zustand **TRUE**.

B.6.2. Mehrfach-Verbindungen

Der boolesche Zustand einer horizontalen Verbindungslinie ist in ihren linken und rechten Enden der gleiche. Mit kombinierten, horizontalen und vertikalen Verbindungen können **Mehrfach-Verbindungen** konstruiert werden. Der Zustand der Enden in einer Mehrfach-Verbindung respektiert logische Regeln.

Eine **Mehrfach-Verbindung nach links** kombiniert **mehrere** horizontale Verbindungslinien, die **links** an eine vertikale Verbindungslinie angeschlossen werden, sowie **eine einzige** horizontale Verbindungslinie **rechts**. Der Zustand des rechten Endes ist das Ergebnis des **LOGISCHEN ODER** zwischen den Zuständen der linken Enden.

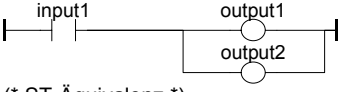
(* Beispiel einer Mehrfach-Verbindung nach links *)



(* Zustand des rechten Endes : (v1 OR v2 OR v3) *)

Eine **Mehrfach-Verbindung nach rechts** kombiniert **eine einzige** horizontale Verbindungslinie, die **links** an eine vertikale Verbindungslinie angeschlossen ist, sowie **mehrere** horizontale Verbindungslinien **rechts**. Der Zustand des linken Endes wird in sämtliche rechten Enden propagiert.

(* Beispiel einer Mehrfach-Verbindung nach rechts *)

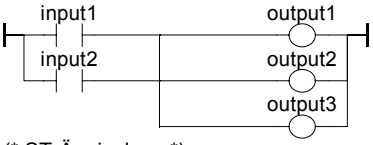



```
(* ST-Äquivalenz *)
output1 := input1;
output2 := input1;
```

Eine **Mehrfach-Verbindung links und rechts** kombiniert **mehrere** horizontale Verbindungslinien **links**, die an eine vertikale Verbindungslinie angeschlossen werden, sowie

mehrere horizontale Verbindungen **rechts**. Der Zustand sämtlicher rechten Enden ist das Ergebnis des **LOGISCHEN ODER** zwischen den Zuständen der linken Enden.

(* Beispiel einer Mehrfach-Verbindung links und rechts *)




```
(* ST-Äquivalenz *)
output1 := input1 ODER input2;
output2 := input1 ODER input2;
output3 := input1 ODER input2;
```

B.6.3. Kontakte und Spulen

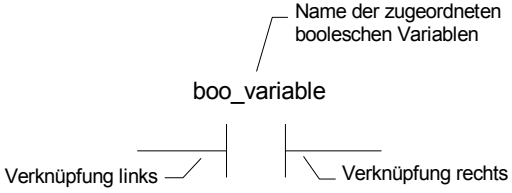
Folgende Kontakt-Typen können in einem Kontaktplan verwendet werden :

- Direkter Kontakt
- Invertierter Kontakt
- Kontakt mit Flankenerkennung

Folgende Spulen-Typen können in einem Kontaktplan verwendet werden :

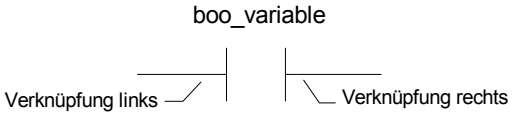
- Direkte Spule
- Invertierte Spule
- Spule mit SETZE-Forcierung
- Spule mit RÜCKSETZE-Forcierung
- Spule mit Flanken-Erkennung

Der Name der zugeordneten Variablen wird über das Symbol geschrieben :



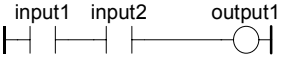
B.6.3.1. Direkte Kontakte

Ein direkter Kontakt stellt eine Operation zwischen dem Zustand einer **Verbindungslinie** und einer booleschen **Variablen** dar.



Der Zustand der Verbindung rechts ist das Ergebnis des **LOGISCHEN UND** zwischen dem Zustand der linken Verbindung und dem Wert der Variablen, die dem Kontakt zugeordnet ist.

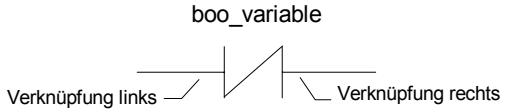
(* Beispiel direkter Kontakte *)




```
(* ST-Äquivalenz *)
output1 := input1 AND input2;
```

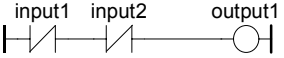
B.6.3.2. Invertierte Kontakte

Ein invertierter Kontakt stellt eine Operation zwischen dem Zustand einer **Verbindungslinie** und der **logischen Inversion** einer booleschen **Variablen** dar.



Der Zustand der Verbindung rechts ist das Ergebnis des **LOGISCHEN UND** zwischen dem Zustand der linken Verbindung und der **LOGISCHEN INVERSION** des Werts der Variablen, die dem Kontakt zugeordnet ist.

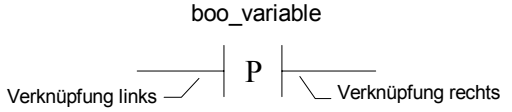
(* Beispiel invertierter Kontakte *)




```
(* ST-Äquivalenz *)
output1 := NOT (input1) AND NOT (input2);
```

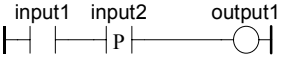
B.6.3.3. Kontakte mit Erkennung positiver Flanken

Diese Kontakte stellen eine Operation zwischen dem Zustand einer **Verbindungslinie** und dem Übergang einer booleschen **Variablen** zu dem Zustand **TRUE** dar.



Die Verbindung rechts nimmt den Zustand **TRUE** an, wenn der Zustand der Verbindung links **TRUE** ist und wenn der Zustand der zugeordneten Variablen von **FALSE** zu **TRUE** übergeht. In allen anderen Fällen nimmt sie den Zustand **FALSE** an.

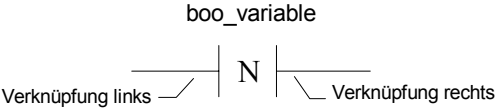
(* Kontakt mit Erkennung einer positiven Flanke *)




```
(* ST-Äquivalenz *)
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev ist der Wert von input2 im vorherigen Zyklus *)
```

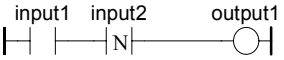
B.6.3.4. Kontakte mit Erkennung negativer Flanken

Diese Kontakte stellen eine Operation zwischen dem Zustand einer **Verbindungslinie** und dem Übergang einer booleschen **Variablen** zu dem Zustand **FALSE** dar.



Die rechten Verbindung nimmt den Zustand **TRUE** an, wenn der Zustand der Verbindung links **TRUE** ist und wenn der Zustand der zugeordneten Variablen von **TRUE** zu **FALSE** übergeht. In allen anderen Fällen nimmt sie den Zustand **FALSE** an.

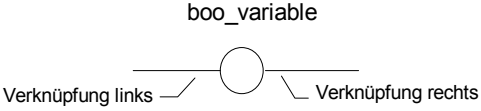
(* Kontakt mit Erkennung einer negativen Flanke *)




```
(* ST-Äquivalenz *)
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev ist der Wert von input2 im vorherigen Zyklus *)
```

B.6.3.5. Direkte Spulen

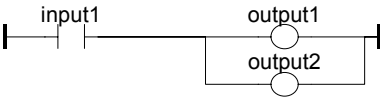
Eine direkte Spule ermöglicht die Forcierung eines **booleschen Ausgangs** mit dem Zustand einer **Verbindungslinie**.



Die zugeordnete Variable wird zu dem **Zustand der linken Verbindung** forciert. Der Zustand der Verbindung links wird in die Verbindung rechts propagiert. Die rechte Verbindung kann an

die rechte Stromschiene angeschlossen werden. Die zugeordnete, boolesche Variable muß eine **AUSGANGS**- oder **INTERNE** Variable sein. Der Name der zugeordneten Variablen kann der Name eines bearbeiteten Programms sein (nur für **Unterprogramme**). In diesem Fall stellt die Spule die Zuordnung des vom Unterprogramm zurückgegebenen Werts dar.

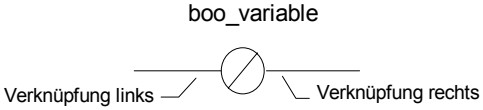
(* Beispiel einer direkten Spule *)




```
(* ST-Äquivalenz: *)
output1 := input1;
output2 := input1;
```

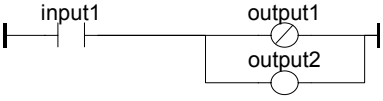
B.6.3.6. Invertierte Spulen

Eine invertierte Spule ermöglicht die Forcierung eines **booleschen Ausgangs** zu einer **logischen Inversion** des Zustands einer **Verbindungslinie**.



Die zugeordnete Variable wird zu einer **logischen Inversion** des **Zustands der linken Verbindung** forciert. Der Zustand der linken Verbindung wird in die rechte Verbindung propagiert. Die rechte Verbindung kann an die rechte Stromschiene angeschlossen werden. Die zugeordnete, boolesche Variable muß eine **AUSGANGS**- oder **INTERNE** Variable sein. Der Name der zugeordneten Variablen kann der Name eines bearbeiteten Programms sein (nur für **Unterprogramme**). In diesem Fall stellt die Spule die Zuordnung des vom Unterprogramm zurückgegebenen Werts dar.

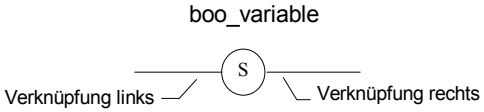
(* Beispiel einer invertierten Spule *)




```
(* ST-Äquivalenz: *)
output1 := NOT (input1);
output2 := input1;
```

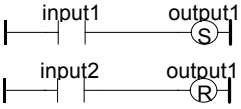
B.6.3.7. "SETZE"-Spulen

Eine Spule mit "Setze"-Aktion ermöglicht das Forcieren eines **booleschen Ausgangs** zu dem Zustand der **Verbindungslinie**.



Die zugeordnete Variable wird **ZU TRUE FORCIERT**, wenn die **Verbindung links** den Zustand TRUE annimmt. Die Variable behält diesen Wert, bis ein entgegengesetzter Befehl von einer Spule des Typs "RÜCKSETZE" gegeben wird. Der Zustand der linken Verbindung wird in die rechte Verbindung propagiert. Die rechte Verbindung kann an die rechte Stromschiene angeschlossen werden. Die zugeordnete, boolesche Variable muß eine **AUSGANGS**- oder **INTERNE** Variable sein.

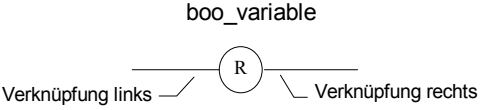
(* Beispiel einer Spule mit SETZE- und RÜCKSETZE-Aktionen *)




```
(* ST-Äquivalenz: *)
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

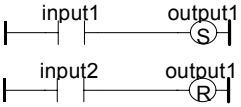
B.6.3.8. "RÜCKSETZE"-Spule

Eine Spule mit "Rücksetze"-Aktion ermöglicht das Forcieren eines **booleschen Ausgangs** zu dem Zustand einer **Verbindungslinie**.



Die zugeordnete Variable wird **ZU FALSE FORCIERT**, wenn die **Verbindung links** den Zustand TRUE annimmt. Die Variable behält diesen Wert, bis ein entgegengesetzter Befehl von einer Spule des Typs "SETZE" gegeben wird. Der Zustand der linken Verbindung wird in die rechte Verbindung propagiert. Die rechte Verbindung kann an die rechte Stromschiene angeschlossen werden. Die zugeordnete, boolesche Variable muß eine **AUSGANGS**- oder **INTERNE** Variable sein.

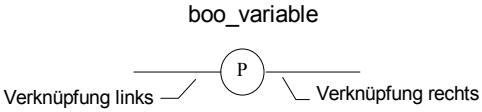
(* Beispiel einer Spule mit SETZE- und RÜCKSETZE-Aktionen *)




```
(* ST-Äquivalenz: *)
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

B.6.3.9. Spulen mit Erkennung steigender Flanken

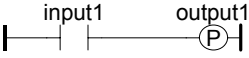
"Positive"Spulen ermöglichen den **booleschen Ausgang** des booleschen Zustands einer **Verbindungslinie**. Diese Art von Spule ist nur im Schnellen KOP-Editor verfügbar.



Die zugeordnete Variable wird zu **TRUE** forciert, wenn der boolesche **Zustand der linken Verbindung** von FALSE auf TRUE steigt. In allen anderen Fällen wird die Ausgangsvariable auf FALSE zurückgesetzt. Der Zustand der linken Verbindung wird in die rechte Verbindung propagiert. Rechte Verbindungen können an die rechte vertikale Stromschiene angeschlossen werden.

Die zugeordnete boolesche Variable muß eine **AUSGANG**s- oder **INTERNE** Variable sein.

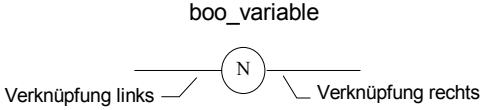
(* Anwendungsbeispiel einer "positiven" Spule *)




```
(* ST-Äquivalenz: *)
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev ist der Wert von input1 im vorherigen Zyklus *)
```

B.6.3.10. Spulen mit Erkennung fallender Flanken

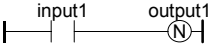
"Negative" Spulen ermöglicht den **booleschen Ausgang** des booleschen Zustands einer **Verbindungslinie**. Diese Art von Spule ist nur im Schnellen KOP-Editor verfügbar.



Die zugeordnete Variable wird zu **TRUE** forciert, wenn der boolesche **Zustand der linken Verbindung** von TRUE auf FALSE fällt. In allen anderen Fällen wird die Ausgangsvariable auf FALSE zurückgesetzt. Der Zustand der linken Verbindung wird in die rechte Verbindung propagiert. Rechte Verbindungen können an die rechte vertikale Stromschiene angeschlossen werden.

Die zugeordnete boolesche Variable muß eine **AUSGANG**s- oder **INTERNE** Variable sein.

(*Anwendungsbeispiel einer "positiven" Spule *)



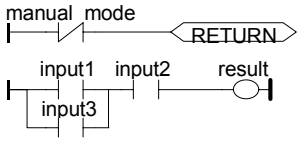

```
(* ST-Äquivalenz: *)
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev ist der Wert von input1 im vorherigen Zyklus *)
```

B.6.4. RETURN-Anweisung

Das **RETURN-**Symbol kann benutzt werden, um die bedingte Beendung einer Programmausführung darzustellen. Rechts vom RETURN-Symbol darf keine Verbindung angebracht werden.

Wenn der Zustand der **linken Verbindung TRUE** ist, wird das weitere KOP-Diagramm (darauffolgende Zeilen) nicht interpretiert.
Anmerkung: Wenn das KOP-Programm ein Unterprogramm ist, muß sein Name einer Ausgangsspule zugeordnet werden, um den Rückgabewert zu setzen (Rückgabe an das aufrufende Programm).

(* Anwendungsbeispiel des RETURN-Symbols *)




```
(* ST-Äquivalenz *)
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```

B.6.5. Sprünge und Marken

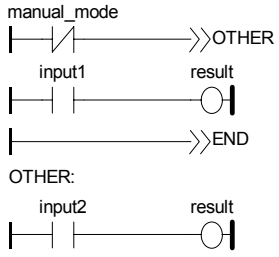
Marken und bedingte oder unbedingte Sprünge werden benutzt, um die Diagrammausführung zu steuern. Rechts von einem Marken- oder Sprungssymbol kann kein Anschluß vorgenommen werden.

Die folgenden Notationen werden benutzt:

>>LAB ..... Sprung zu Marke mit dem Namen "LAB"
LAB: ..... Definition der Marke mit dem Namen "LAB"

Wenn die Verbindung links vom Symbol den Zustand **TRUE** annimmt, wird die Programmausführung zur entsprechenden Marke umgeleitet.

(* Anwendungsbeispiel für Sprünge und Marken *)

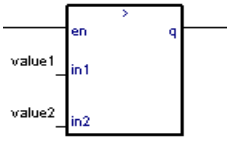



```
(* AWL-Äquivalenz *)
                ldn         manual_mode
                jmpc        other
                ld          input1
                st          result
                jmp         END
OTHER:          ld          input2
                st          result
END:            (*Programmende *)
```

B.6.6. Funktionsbausteine im KOP

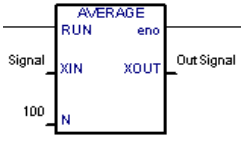
Im Schnellen KOP-Editor werden Funktionsbausteine mit booleschen Linien verbunden. Eine Funktion kann in Wirklichkeit ein Operator, ein Funktionsbaustein oder eine Funktion sein. Da alle Bausteine nicht immer einen booleschen Eingang oder Ausgang besitzen, bewirkt das Einfügen solcher Bausteine in ein KOP-Diagramm die Verwendung zusätzlicher EN- und ENO-Parameter in der Baustein-Schnittstelle. Im FBS/KOP-Editor werden die EN- und ENO-Parameter nicht benutzt, da die Variablen mit den erforderlichen Typen verbunden werden können.

• Der "EN"-Eingang
Bei manchen Operatoren, Funktionen und Funktionsbausteinen ist der erste Eingang nicht immer boolesch. Da der erste Eingang immer an das Rung angeschlossen werden muß, wird automatisch ein anderer Eingang mit dem Namen "**EN**" an der ersten Stelle eingefügt. Der Baustein wird nur ausgeführt, wenn der **EN**-Eingang TRUE ist. Das folgende Beispiel zeigt einen Vergleichs-Operatoren und den äquivalenten Code in ST:



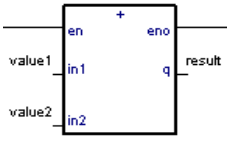

```
IF rung_state THEN
   q := (value1 > value 2);
ELSE
   q := FALSE;
END_IF;
(*Rung mit q-Status fortführen*)
```

• Der "ENO"-Ausgang
Bei manchen Operatoren, Funktionen und Funktionsbausteinen ist der erste Ausgang nicht immer boolesch. Da der erste Ausgang immer an das Rung angeschlossen werden muß, wird automatisch ein anderer Ausgang mit dem Namen "**ENO**" an der ersten Stelle eingefügt. Der **ENO**-Ausgang nimmt immer den gleichen Zustand wie der erste Eingang des Bausteins an. Das folgende Beispiel zeigt einen AVERAGE-Funktionsbaustein und den äquivalenten Code in ST:




```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* Rung mit eno-Status fortführen *)
```

• Die "EN"- und "ENO"-Parameter
In manchen Fällen sind **EN** und **ENO** erforderlich. Es folgt das Beispiel eines arithmetischen Operators und der äquivalente Code in ST:




```
IF rung_state THEN
   result := (value1 + value2);
END_IF;
eno := rung_state;
(*Rung mit eno-Status fortführen *)
```

B.7. ST-Sprache

Die **ST**-Sprache (Strukturierter Text) ist eine hohe Textsprache, die für Automatisierungsapplikationen bestimmt ist. Sie wird hauptsächlich für die Beschreibung von komplexen Prozeduren benutzt, die mit den graphischen Sprachen schwer darzustellen sind. Die ST-Sprache ist die vorgegebene Sprache für die Programmierung der Schrittaktionen und der Bedingungen, die den Transitionen in der **AS**-Sprache zugeordnet werden.

B.7.1. ST-Syntax

Ein ST-Programm besteht aus einer Folge von **Anweisungen**, die jeweils mit einem Semikolon enden (**";"**). Die im Quellcode benutzten Namen (Bezeichner von Variablen, Konstanten, Sprach-Schlüsselwörtern...) werden durch **passive Trennzeichen** (Leerzeichen, Zeilenende oder Tab) oder **aktive Trennzeichen** getrennt, die bestimmte Operationen darstellen. Kommentare können in der Programmierung beliebig eingefügt werden. Ein Kommentar beginnt mit den Zeichen **"(*"** und endet mit den Zeichen **"*)"**. Jede Anweisung endet mit einem Semikolon (**";"**). In der ST-Sprache gibt es die folgenden Standard-Anweisungen :

- **Zuweisung** (Variable := Ausdruck;)
- Aufruf eines **Unterprogramms** oder einer **Funktion**
- Aufruf eines **Funktionsbausteins**
- **Selektion**sanweisungen (IF, THEN, ELSE, CASE)
- **Wiederholung**sanweisungen (FOR, WHILE, REPEAT)
- **Kontroll**anweisungen (RETURN, EXIT)
- Spezialanweisungen für die Verbindung mit der **AS-**Sprache.

Passive Trennzeichen dürfen beliebig zwischen den aktiven Trennzeichen, konstanten Ausdrücken und Bezeichnern eingefügt werden. Passive Trennzeichen sind **Leerzeichen** und die Zeichen **Tabulation** und **Zeilenende.** Im Gegensatz zu zeilen-formatierten Sprachen, wie die **AWL**, können die Zeilenendzeichen überall im Programm verwendet werden. Bei der Benutzung von passiven Trennzeichen sollten Sie die folgenden Regeln beachten, damit der Quellcode leicht lesbar ist :

- Schreiben Sie nicht mehrere Anweisungen auf einer Zeile.
- Benutzen Sie Tabulationen, um die Kontrollstrukturen einzurücken.
- Fügen Sie Kommentare ein.

Lesbarkeit des Quellcodes - Beispiele:

Schlechte Lesbarkeit	Gute Lesbarkeit

imax := max_ite; cond := X12;
if not(cond (* alarm *)
then return; end_if;
for i (* index *) := 1 to max_ite
do if i <> 2 then Spcall();
end_if; end_for;
(* no effect if alarm *)
	(* imax : number of iterations *)
	(* i: FOR statement index *)
	(* cond: process validity *)
	imax := max_ite;
	cond := X12;
	if not (cond) then
	  return;
	end_if;
	
	(* process loop *)
	for i := 1 to max_ite do
	  if i <> 2 then
	  Spcall ();
	  end_if;
	end_for;

B.7.2. Ausdrücke

ST-Ausdrücke kombinieren variable oder konstante **Operatoren** und **Operanden**. Für jeden einfachen Ausdruck (der Operanden mit einem ST-Operator kombiniert) müssen die Operanden-**Typen** konsistent sein. Ein einfacher Ausdruck ist vom Typ seiner Operanden und des verwendeten Operatoren und kann in einem komplizierteren Ausdruck verwendet werden. Zum Beispiel :

(boo_var1 AND boo_var2)	ist vom Typ BOOLESCH
not (boo_var1)	ist vom Typ BOOLESCH
(sin (3.14) + 0.72)	ist vom Typ analog, REAL
(t#1s23 + 1.78)	ist ein ungültiger Ausdruck

Man benutzt **Klammern**, um die einzelnen Teile eines Ausdrucks zu trennen und die Bewertungsordnung expliziert darzustellen. Werden keine Klammern eingefügt, so wird die Bewertungsordnung durch die **Priorität** zwischen den ST-Operatoren bestimmt. Zum Beispiel :

2 + 3 * 6	gleich 2+18=20	weil die Multiplikation prioritär ist
(2+3) * 6	gleich 5*6=30	Priorität durch die Klammern

Achtung: Es können nicht mehr als **8** Klammer-Ebenen in einem Ausdruck verschachtet werden.

B.7.3. Aufrufe an Funktionen und Funktionsbausteine

Die Aufrufbestimmungen der ST-Sprache für die Funktionen betrifft die folgenden Objekte :

- Unterprogramme

- In den IEC-Sprachen geschriebene Bibliotheksfunktionen und -Funktionsbausteine
- C-Funktionen und -Funktionsbausteine
- Typ-Umrechnungsfunktionen

B.7.3.1. Aufruf von Unterprogrammen oder Funktionen

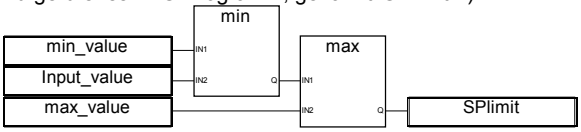
Name:	Name des aufgerufenen Unterprogramms oder der Bibliotheksfunktion(in den IEC-Sprachen oder in "C" geschrieben)
Bedeutung:	ruft ST-, AWL-, KOP- oder FBS-Unterprogramm oder -Funktion oder eine "C"-Funktion auf und bezieht ihren Rückgabewert
Syntax:	**<var_ana> := <uprog> (<par1>,** ... **<parN>);**
Operanden:	Der Rückgabewert und die gegebenen Werte müssen die Parametertypen respektieren
Rückgabe:	Der vom Unterprogramm zurückgegebene Wert

Der Aufruf eines Unterprogramm kann in einem komplexen Ausdruck oder in einer AS-Bedingung verwendet werden.

Beispiel 1: Aufruf eines Unterprogramms

```
(*ST-Hauptprogramm *)
(* bezieht einen analogen Wert und rechnet ihn in einen begrenzten Zeitwert um *)
ana_timeprog := SPlimit ( tprog_cmd );
appl_timer := tmr (ana_timeprog * 100);
```

(* Aufgerufenes FBS-Programm, genannt 'SPlimit' *)



Beispiel 2: Aufruf einer Funktion

```
(*In komplexen Ausdücken benutzte Funktionen: min, max, right, mlen und left sind Standard-
"C"-Funktionen *)
limited_value := min (16, max (0, input_value) );
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

B.7.3.2. Aufruf von Funktionsbausteinen

Name:	Name der Instance des Funktionsbausteins
Funktion:	Ruft einen Funktionsbaustein der ISaGRAF Bibliothek auf und greift auf seine Returnparameter zu
Syntax:	**<name_bau> (<pa1>, <pa2>** ... **);** **<ergebnis> := <name_bau>. <param_ret_1>;** ... **<ergebnis> := <name_bau>. <name_ret_N>;**

Operanden:	Die Aufrufparameter und die Zuordnung der Returnparameter müssen die in der Schnittstelle der Funktion definierten Typen respektieren
Rückgabewert:	Siehe Syntax, um die Returnparameter zu erhalten.

Siehe ISaGRAF Bibliothek über die Rolle und die genaue Schnittstelle eines jeden Funktionsbausteins. Die Instance des Funktionsbausteins (Name der Kopie) muß im Datenverzeichnis deklariert werden.

Beispiel :

```
(* ST-Programm mit Aufruf eines "C"-Funktionsbausteins *)
(* Deklaration der Instance im Datenverzeichnis: *)
(* trigb1 : bloc R_TRIG - Erkennung einer steigenden Flanke *)

(* Aktivierung des Funktionsbausteins in ST  *)
trigb1 (b1);
(* Zugriff zu den Returnparametern *)
If (trigb1.Q) Then nb_fronts := nb_fronts + 1; End_if;
```

B.7.4. Spezifische boolesche ST-Operatoren

Die folgenden booleschen Operatoren sind spezifisch zur ST-Sprache:

- REDGE — Erkennung einer steigenden Flanke
- FEDGE — Erkennung einer fallenden Flanke

Andere boolesche Standard-Operatoren, so wie:
- NOT — boolesche Negation
- AND (&) — logisches UND
- OR — logisches ODER
- XOR — logisches exklusiv-ODER

können benutzt werden. Sie werden im Abschnitt 'Standard-Operatoren, Funktionsbausteine und Funktionen' beschrieben.

B.7.4.1. "REDGE"-Operator

Name:	**REDGE**
Funktion:	bewertet die steigende Flanke eines komplexen, booleschen Ausdrucks
Syntax:	**<flanke> := REDGE (<ausdruck_boo>, <variable_speich>);**
Operanden:	Der erste Operand ist eine boolesche Variable oder ein komplexer, boolescher Ausdruck. Der zweite Operand ist eine boolesche, interne Variable, die benutzt wird, um den vorherigen Zustand des verarbeiteten Ausdrucks zu speichern.
Rückgabe:	TRUE nur dann, wenn der Zustand des Ausdrucks von FALSE auf TRUE übergeht FALSE in allen anderen Fällen

Die steigende Flanke eines Ausdrucks kann nur ein einziges Mal mit der REDGE-Funktion im selben Ausführungszyklus erkannt werden.

Achtung: Die "Speichervariable", die benutzt wird, um den vorherigen Zustand des Ausdrucks zu speichern, darf nicht in Flanken-Berechnungen anderer Ausdrücke verwendet werden.

Wenn der dem Kontakt zugeordnete Ausdruck eine boolesche Variable mit Namen "**xxx**" ist, deklariert man eine interne Variable und nennt sie "**EDGE_xxx**" für die REDGE- und FEDGE-Operationen, die diese Variable betreffen. Hierdurch werden Konflikte zwischen den Variablen bei anderen Flankenberechnungen vermieden.

Beispiel :

```
(* ST-Programm, das einen REDGE-Operator benutzt *)
(*dieses Programm zählt die steigenden Flanken eines bool. Eingangs*)
(* Bi120 ist der Name der Eingangsvariablen *)
(* Edge_Bi120 ist die Variable, die für das Speichern des Zustands von Bi120 benutzt wird *)
If REDGE (Bi120, Edge_Bi120) Then
      Counter := Counter + 1;
End_if;
```

Anmerkung: Dieser Operator ist nicht Teil der IEC1131-3 Norm. Vielleicht bevorzugen Sie den Standard-Baustein R_TRIG. Er wurde aus Kompatibilitätsgründen erhalten.

B.7.4.2. Operator "FEDGE"

Name:	**FEDGE**
Funktion:	bewertet die fallende Flanke eines komplexen, booleschen Ausdrucks
Syntax:	**<flanke> := FEDGE (<ausdruck_boo>, <variable_memo>);**
Operanden:	Der erste Operand ist eine boolesche Variable oder ein komplexer, boolescher Ausdruck. Der zweite Operand ist eine boolesche, interne Variable, die benutzt wird, um den vorherigen Zustand des verarbeiteten Ausdrucks zu speichern.
Rückgabe:	TRUE nur dann, wenn der Zustand des Ausdrucks von TRUE auf FALSE übergeht FALSE in allen anderen Fällen

Die fallende Flanke eines Ausdrucks kann nur ein einziges Mal mit der FEDGE-Funktion im selben Ausführungszyklus erkannt werden.

Achtung: Die "Speichervariable", die benutzt wird, um den vorherigen Zustand des Ausdrucks zu speichern, darf nicht in Flanken-Berechnungen anderer Ausdrücke verwendet werden.

Wenn der dem Kontakt zugeordnete Ausdruck eine boolesche Variable mit Namen "**xxx**" ist, deklariert man eine interne Variable und nennt sie "**EDGE_xxx**" für die REDGE- und FEDGE-Operationen, die diese Variable betreffen. Hierdurch werden Konflikte zwischen den Variablen bei anderen Flankenberechnungen vermieden.

Beispiel :

```
(* ST-Programm, das einen FEDGE-Operator benutzt *)
(* dieses Programm zählt die fallenden Flanken eines booleschen Eingangs*)
(* Bi120 ist der Name der Eingangsvariablen *)
(* Edge_Bi120 ist die Variable, die für das Speichern des Zustands von Bi120 benutzt wird
      *)
If FEDGE (Bi120, Edge_Bi120) Then
      Counter := Counter + 1;
End_if;
```

Anmerkung: Dieser Operator ist nicht Teil der IEC1131-3 Norm. Vielleicht bevorzugen Sie den Standard-Baustein F_TRIG. Er wurde aus Kompatibilitätsgründen erhalten.

B.7.5. Anweisungen der ST-Sprache

Hier die verschiedenen Anweisungstypen der ST-Sprache :

- Zuweisung
- Anweisung RETURN
- Selektion IF-THEN-ELSIF-ELSE
- Anweisung CASE
- Wiederholung WHILE
- Wiederholung REPEAT
- Wiederholung FOR
- Anweisung EXIT

B.7.5.1. Zuweisung

Name:	**:=**
Funktion:	weist einer Variablen einen Ausdruck zu
Syntax:	**<variable> := <ausdruck> ;**
Operanden:	die Variable muß eine interne oder Ausgangsvariable sein die Variable und der Ausdruck müssen vom gleichen Typ sein

Der Ausdruck kann ein Aufruf an ein Unterprogramm oder eine Funktion der ISaGRAF Bibliothek sein.

Beispiel :
(* ST-Programm - Zuweisungsbeispiel *)

```
(* Variable <<= Variable *)
bo23 := bo10;

(* Variable <<= Ausdruck *)
bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;
```

```
(* mit Unterprogramm-Aufruf *)
rc := PSelect ( );

(* mit Aufruf einer Funktion  *)
limited_value := min (16, max (0, input_value) );
```

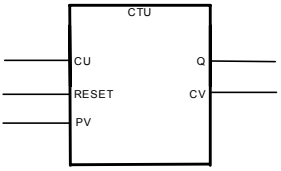
B.7.5.2. "RETURN"-Anweisung

Name: **RETURN**
Funktion: Beendet die Programm-Ausführung
Syntax: **RETURN ;**
Operanden: (keine)

Wenn die RETURN-Anweisung in einem AS-Aktionsblock erscheint, betrifft die Beendung nur diesen Aktionsblock.

Beispiel :

(* FBS-Spezifikationen des Programms: programmierbarer Zähler *)



(* St-Implementierung des Programms, mit RETURN-Anweisung *)

```
If not (CU) then
      Q := false;
      CV := 0;
      RETURN; (* beendet das Programm *)
end_if;

if R then
      CV := 0;
else
      if (CV < PV) then
            CV := CV + 1;
      end_if;
end_if;
Q := (CV >= PV);
```

B.7.5.3. "IF-THEN-ELSE"-Anweisung

Name: **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**
Funktion: Führt eine von zwei ST-Anweisunglisten aus.
Die Wahl erfolgt entsprechend des Zustands eines booleschen Ausdrucks
Syntax: **IF** <**bool_ausdruck**> **THEN**

```
   <anweisung> ;
   <anweisung> ;
   ...
ELSIF <bool_ausdruck> THEN
   <anweisung> ;
   <anweisung> ;
   ...
ELSE
   <anweisung> ;
   <anweisung> ;
   ...
END_IF;
```

Die ELSE- und ELSIF-Anweisungen sind optionell. Wird ELSE weggelassen, wird keine Anweisung ausgeführt, wenn der Selektionsausdruck den Wert FALSE annimmt.
Beispiel :

```
(* ST-Programm mit IF-Anweisung *)
IF manual AND not (alarm) THEN
      level := manual_level;
      bx126 := bi12 OR bi45;
ELSIF over_mode THEN
      level := max_level;
ELSE
      level := (lv16 * 100) / scale;
END_IF;
(* Struktur IF ohne ELSE *)
If overflow THEN
       alarm_level := true;
END_IF;
```

B.7.5.4. "CASE"-Anweisung

Name: **CASE ... OF ... ELSE ... END_CASE**

Funktion: Führt eine von mehreren ST-Anweisungslisten aus. Die Selektion findet im Einverständnis mit dem Wert eines analogen, ganzzahligen Ausdrucks statt.

Syntaxe:

```
CASE <ganzzahliger_ausdruck> OF
<wert> : <anweisungen> ;
<wert> , < wert > : < anweisungen > ;
...
ELSE
< anweisungen> ;
END_CASE;
```

Die Selektionswerte müssen konstante, ganzzahlige Ausdrücke sein. Mehrere, durch Kommas getrennte Selektionswerte können zu derselben Anweisungsliste leiten. Die ELSE-Anweisung ist optionnell.

Beispiel:

```
(* ST-Programm mit CASE-Anweisung *)
CASE error_code OF
      255:     err_msg := 'Divide by zero';
               fatal_error := TRUE;
      1:       err_msg := 'Overflow';
      2, 3:    err_msg := 'Bad sign';
ELSE
      err_msg := 'Unknown error';
END_CASE;
```

B.7.5.5. "WHILE"-Anweisung

Name: **WHILE ... DO ... END_WHILE**
Funktion: Wiederholungsanweisung mit Wiederholungstest am Anfang der Schleife.
Syntax:

```
WHILE <bool_ausdruck> DO
   <anweisung> ;
   <anweisung> ;
   ...
END_WHILE ;
```

Achtung: Da ISaGRAF ein **synchrones** System ist, werden die Eingangsvariablen während der Wiederholung einer WHILE-Schleife nicht aktualisiert. Die Entwicklung einer Eingangsvariablen darf nie in der Bedingung einer WHILE-Struktur getestet werden.

Beispiel :

```
(* ST-Programm mit WHILE-Anweisung *)
(* dieses Programm benutzt spezifische "C"-Funktionen, um die Zeichen *)
(* auf einer seriellen Kommunikations-Schnittstelle zu lesen          *)
string := ''; (* leere Kette *)
nbchar := 0;
WHILE ((nbchar < 16) & ComIsReady ( )) DO
       string := string + ComGetChar ( );
       nbchar := nbchar + 1;
END_WHILE;
```

B.7.5.6. "REPEAT"-Anweisung

Name: **REPEAT ... UNTIL ... END_REPEAT**
Funktion: Wiederholungsanweisung mit Wiederholungstest am Ende der Schleife.
Syntax:

```
REPEAT
   <anweisung> ;
   <anweisung> ;
  ...
UNTIL <bool-bedingung>
END_REPEAT ;
```

Achtung: Da ISaGRAF ein **synchrones** System ist, werden die Eingangsvariablen während der Wiederholungen einer REPEAT-Schleife nicht aktualisiert. Die Entwicklung einer Eingangsvariablen darf nie in der Bedingung einer REPEAT-Struktur getestet werden.

Beispiel :

```
(* ST-Programm mit REPEAT-Anweisung *)
(* dieses Programm benutzt spezifische "C"-Funktionen, um die Zeichen *)
(* auf einer seriellen Kommunikations-Schnittstelle zu lesen          *)
string := ''; (* leere Kette *)
nbchar := 0;
IF ComIsReady ( ) THEN
      REPEAT
            string := string + ComGetChar ( );
            nbchar := nbchar + 1;
      UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) );
      END_REPEAT;
END_IF;
```

B.7.5.7. "FOR"-Anweisung

Name:	**FOR ... TO ... BY ... DO ... END_FOR**	
Funktion:	Wiederholungsstruktur, die eine Index-Variable vom Typ ganzzahlig, analog benutzt	
Syntax:	**FOR <index> := <mini> TO <maxi> BY <schritt>** **DO** **<anweisung> ;** **<anweisung> ;** **END_FOR;**	
Operanden:	**index**:	bei jeder Wiederholung inkrementierte interne, analoge Variable
	mini:	Initialwert der Index-Variablen (Wert bei der ersten Wiederholung)
	maxi:	zugelassener Maximalwert für den Index
	schritt:	Inkrementierungs-Schritt des Index (bei jeder Wiederholung)

Achtung: Da ISaGRAF ein **synchrones** System ist, werden die Eingangsvariablen während den Wiederholungen einer FOR-Schleife nicht aktualisiert.

Die [BY schritt] Anweisung ist optionell. Wenn nicht spezifiert, ist die Vorgabe für den Inkrementierungs-Schritt 1. Hier das Äquivalent "While" einer FOR-Anweisung :

```
index := mini;
while (index <= maxi) do
      <anweisung> ;
      <anweisung> ;
      index := index + schritt;
end_while;
```

Beispiel :

```
(* ST-Programm mit FOR-Anweisung *)
(* Dieses Programm entnimmt einer Kette die "Zahlen"-Zeichen *)
length := mlen (message);
target := ''; (* leere Kette *)
FOR index := 1 TO length BY 1 DO
      code := ascii (message, index);
      IF (code >= 48) & (code <= 57) THEN
            target := target + char (code);
      END_IF;
END_FOR;
```

B.7.5.8. "EXIT"-Anweisung

Name:	**EXIT**
Funktion:	verläßt eine der Wiederholungsstrukturen FOR, WHILE oder REPEAT
Syntaxe:	**`EXIT;`**

Die EXIT-Anweisung wird im allgemeinen in einer IF-Selektionanweisung benutzt.

Beispiel:

```
(* ST-Programm mit EXIT-Anweisung *)
(* dieses Programm sucht ein Zeichen in einer Kette *)
length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
      code := ascii (message, index);
      IF (code  = searched_char) THEN
            found := YES;
            EXIT;
      END_IF;
END_FOR;
```

B.7.6. Erweiterungen der ST-Sprache

Die folgenden Anweisungen und Funktionen sind Erweiterungen der ST-Sprache.

- Timer-Verwaltung: TSTART / TSTOP

Die folgenden Anweisungen ermöglichen einem AS-Vaterprogramm, ein AS-Sohnprogramm zu steuern :

- GSTART startet ein AS-Programm
- GKILL stoppt ein AS-Programm
- GFREEZE hält ein AS-Programm an
- GRST startet ein AS-Programm erneut
- GSTATUS liefert den Status eines AS-Programms

Warnung: Diese Funktionen sind nicht Teil der IEC 1131-3 Norm.
Ein Äquivalent für GSTART und GKILL ist die folgende Syntax im AS-Schritt:

```
child_name(S); (* äquivalent zu GSTART(child_name); *)
child_name(R); (*äquivalent zu GKILL(child_name); *)
```

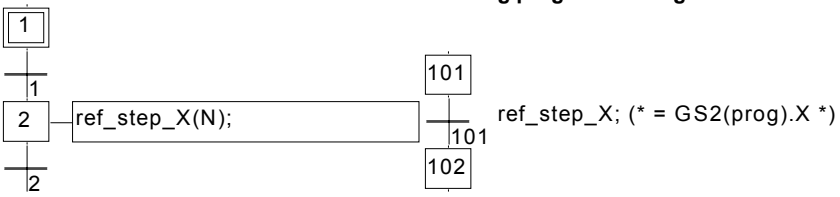
Die folgenden Felder eines AS-Schritts können in Ausdrücken getestet werden :

GSnnn.x ist ein boolescher Wert, der die Schrittaktivität darstellt
GSnnn.t ist ein Zeitwert, der angibt, wie lange der Schritt schon aktiv ist
("**nnn**" ist die Referenznummer des AS-Schritts)

Unter Anwendung der folgenden Syntax ist es außerdem möglich, die Aktivität eines deklarierten Schritts in einem anderen AS-Programm zu testen:

GSnnn(nameprog).x
Warnung: Diese Syntax für die Referenzierung eines Schritts eines anderen Programms ist nicht Teil der IEC 1131-3 Norm. Innerhalb der IEC-Norm kann das gleiche erreicht werden, indem eine globale boolesche Variable im Datenverzeichnis deklariert wird, welche die zu testende Schrittaktivität darstellt (zum Beispiel ref_step_X). Dann fügen Sie die Variable mit dem N-Qualifizierer in den Schritt ein (ref_step_X(N);) und benutzen die Variable in dem Programm, welches die Schrittaktivität testen soll.

Prog program **anderes Programm, das die Schrittaktivität von Prog program benötigt**

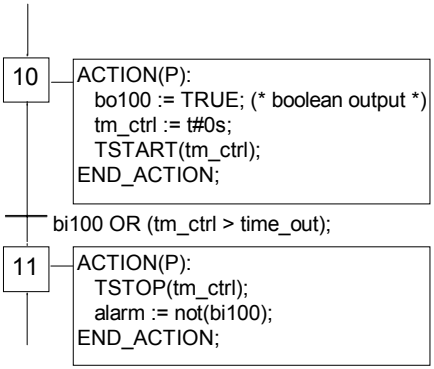


B.7.6.1. "TSTART"-Anweisung

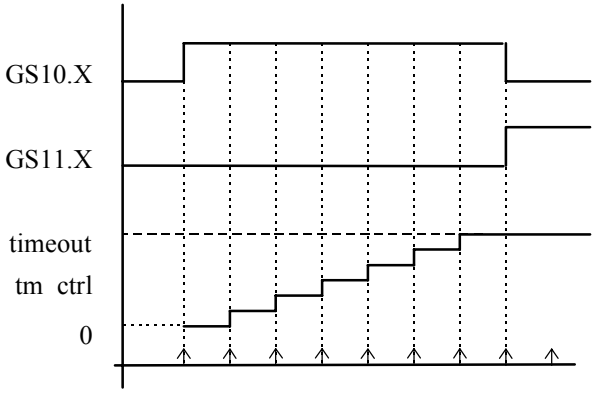
Name: **TSTART**
Funktion: Validiert die automatische Aktualisierung eines Timers. Der aktuelle Wert des Timers wird durch die TSTART-Anweisung nicht verändert
Syntax: `TSTART ( <zeit_variable> );`
Operanden: alle inaktiven Timer-Variablen
Rückgabe: (keine)

Beispiel :

```
(* Anwendungsbeispiel der TSTART- und TSTOP-Anweisungen *)
```



Zeit-Diagramm, wenn bi100 immer FALSE ist:



Der Timer behält einen Zyklus lang den gleichen Wert bei.

B.7.6.2. "TSTOP"-Anweisung

Name:	**TSTOP**
Funktion:	Stoppt die automatische Aktualisierung einer Timer-Variablen. Der Wert der Variablen wird durch den Befehl TSTOP nicht geändert.
Syntax:	**TSTOP (<zeit_variable>);**
Operanden:	jede durch TSTART aktivierte Timer-Variable
Rückgabe:	(keine)

Beispiel :
Siehe TSTART (die Funktion wird weiter oben beschrieben)

B.7.6.3. "GSTART"-Anweisung

Name:	**GSTART**
Funktion:	Startet ein AS-Sohnprogramm, wobei eine Zugriffsberechtigung in jeden seiner Initialisierungsschritte plaziert wird
Syntax:	**GSTART (<sohn_programm>);**
Operanden:	das spezifizierte AS-Programm muß der Sohn des Programms sein, das die GSTART-Anweisung enthält.
Rückgabe:	(keine)

Die Söhne des durch GSTART gestarteten Programms werden nicht automatisch mit ihm gestartet.
Anmerkung: Da GSTART nicht Teil der IEC 1131-3 Norm ist, sollte der S-Qualifizierer bevorzugt werden, mit der folgenden Syntax, um ein AS-Sohnprogramm zu starten:

```
Child_name(S);
```

Beispiel: Anwendung von GSTART und GKILL

```
(* Sequenz 'Sfather' *)
1
Run_cmd;
1
2
ACTION(P):
  GSTART(Schild);
END_ACTION;
NOT (Run_cmd);
2
3
ACTION(P):
  GKILL(Schild);
END_ACTION;
Run_cmd;
3
1
```

```
(* Sequenz 'Schild' *)
1
Bo100;
GS1.t > t#2s;
1
2
Bo101;
GS2.t > t#2s;
2
1
```

B.7.6.4. "GKILL"-Anweisung

Name:	**GKILL**
Funktion:	Stoppt ein AS-Programm, wobei alle Zugriffsberechtigungen in den Schritten gelöscht werden
Syntax:	**GKILL (<sohn-programm>);**
Operanden:	Das spezifizierte AS-Programm muß der Sohn des Programms sein, das die GKILL-Anweisung enthält.
Rückgabe:	(keine)

Die Söhne des gestoppten Programms werden automatisch mit ihm gestoppt.

Anmerkung: Da GKILL nicht Teil der IEC 1131-3 Norm ist, sollte der R-Qualifizierer bevorzugt werden, mit der folgenden Syntax, um einen AS-Sohnprogramm zu stoppen:

```
Child_name(R);
```

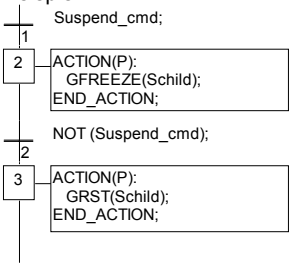
Beispiel: Siehe GSTART (Funktion weiter oben beschrieben)

B.7.6.5. "GFREEZE"-Anweisung

Name:	**GFREEZE**
Funktion:	Löscht alle Zugriffberechtigungen in den Schritten eines Sohnprogramms und speichert deren Position, damit das Programm erneut durch die Anweisung GRST gestartet werden kann.
Syntax:	**GFREEZE (<sohn_programm>);**
Operanden:	Das spezifizierte AS-Programm muß der Sohn des Programms sein, das die GFREEZE-Anweisung enthält.
Rückgabe:	(keine)

Anmerkung: GFREEZE ist nicht Teil der IEC 1131-3 Norm.

Beispiel :



B.7.6.6. "GRST"-Anweisung

Name:	**GRST**
Funktion:	Startet ein AS-Sohnprogramm erneut, wobei alle zuvor durch den Befehl GFREEZE gelöschten Zugriffsberechtigungen wiederhergestellt werden.
Syntax:	**GRST (<sohn_programm>);**
Operanden:	Das spezifizierte AS-Programm muß der Sohn des Programms sein, das die GRST-Anweisung enthält.
Rückgabe:	(keine)

Anmerkung: GRST ist nicht Teil der IEC 1131-3 Norm.

Beispiel: Siehe GFREEZE (Funktion weiter oben beschrieben)

B.7.6.7. "GSTATUS"-Funktion

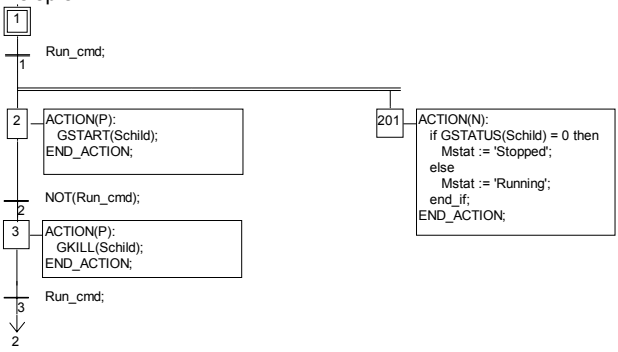
Name:	**GSTATUS**
Funktion:	Liefert den aktuellen Zustand eines AS-Programms.
Syntax:	**<var_ana> := GSTATUS (<sohn_programm>);**
Operanden:	Das spezifizierte AS-Programm muß der Sohn des

Programms sein, das den GSTATUS-Aufruf enthält.

Rückgabe: **0** = das Programm ist **inaktiv** (gestoppt)
1 = das Programm ist **aktiv** (gestartet)
2 = das Programm ist **angehalten**

Anmerkung: GSTATUS ist nicht Teil der IEC 1131-3 Norm.

Beispiel:



B.8. AWL-Sprache

Die **AWL-**Sprache (Anweisungsliste) ist eine Textsprache niedrigen Niveaus. Sie eignet sich besonders für kleinere Anwendungen oder um bestimmte Teile einer Anwendung zu optimieren. Die Anweisungen verarbeiten stets **aktuelle Ergebnisse** (oder **AWL-Verzeichnisse**). Der Operator zeigt an, welche Art von Operation zwischen dem aktuellen Ergebnis und dem Operand erfolgen soll. Das Ergebnis der Operation wird seinerseits im aktuellen Ergebnis gespeichert.

B.8.1. Syntax der AWL-Sprache

Ein AWL-Programm ist eine Liste von **Anweisungen**. Jede Anweisung wird auf einer neuen Zeile geschrieben und muß einen **Operator** enthalten, eventuell vervollständigt durch **Ergänzungen** und, wenn für die Operation erforderlich, einen oder mehrere **Operanden**, die durch Kommas (**','**) getrennt werden. Eine **Marke,** gefolgt von einem Doppelpunkt (**':'**) kann der Anweisung vorangehen. Wenn der Anweisung ein **Kommentar** angehängt wird, muß er das letzte Element der Zeile sein. Ein Kommentar beginnt stets mit dem Zeichen **'(*'** und endet mit **'*)'**. Zwischen den Anweisungen können leere Zeilen eingefügt werden. Kommentare dürfen auch auf Zeilen ohne Anweisungen geschrieben werden. Beispiele von Anweisungszeilen :

MARKE	OPERATOR	OPERAND	KOMMENTARE
Anfang:	**LD**	**IX1**	**(* Druckknopf *)**
	ANDN	**MX5**	**(* Befehl gültig *)**
	ST	**QX2**	**(* startet Motor *)**

B.8.1.1. Marken

Eine **Marke,** gefolgt von einem Doppelpunkt (**':'**) kann der Anweisung vorangehen. Marken können auf Zeilen gesetzt werden, die keine Anweisungen enthalten. Von manchen Anweisungen, wie z.b. den Sprüngen, werden sie als Operanden benutzt. Bei der Benennung der Marken müssen die folgenden Regeln beachtet werden :

- Der Name darf höchstens **16** Zeichen enthalten
- Das erste Zeichen muß ein **Buchstabe** sein.
- Darauffolgende Zeichen können **Buchstaben**, **Zahlen** oder **'_'** sein.

Zwei Marken desselben AWL-Programms dürfen nicht den gleichen Namen tragen. Eine Marke darf den gleichen Namen wie eine Variable haben.

B.8.1.2. Anweisungs-Ergänzungen

Die folgende Liste enthält Ergänzungen, die für die Anweisungen in der AWL-Sprache zugelassen sind. Das Ergänzungszeichen wird dem Anweisungsnamen ohne Trennzeichen beigefügt.

N boolesche Inversion des Operands
(Verzögerungsoperation

C bedingte Operation

Die Ergänzung **'N'** bedeutet, daß der Operand nicht invertiert werden darf, bevor er von der Anweisung verwendet wird. Zum Beispiel wird die Anweisung **ANDN IX12** auf folgende Weise interpretiert :
Ergebnis := Ergebnis AND NOT (IX12)

Die Ergänzung "Klammer auf" **'('** bedeutet, daß die Anweisungsbewertung bis zur nächsten Anweisung verzögert werden muß "Klammer zu" **')'**.

Die Ergänzung **'C'** bedeutet, daß die Anweisung nur dann ausgeführt werden kann, wenn das aktuelle Ergebnis den booleschen Wert TRUE hat (oder ungleich 0 für einen nicht booleschen Wert). Die Ergänzung **'C'** kann mit dem Operator **'N'** kombiniert werden, um anzuzeigen, daß die Anweisung nur dann ausgeführt werden kann, wenn das aktuelle Ergebnis den Wert FALSE hat (oder 0 für eine nicht boolesche Anweisung).

B.8.1.3. Verzögerte Operationen

Da die AWL-Sprache nur ein einziges Verzeichnis verarbeitet (aktuelles Ergebnis), müssen manche Operationen verzögert werden, um die natürliche Ausführungsordnung der Anweisungen zu ändern. Für die Darstellung dieser Verzögerungen werden Klammern benutzt.

'('	ist eine Ergänzung	die Operation wird verzögert
')'	ist eine Anweisung	die verzögerte Operation wird ausgeführt

Die Ergänzung "Klammer auf" **'('** bedeutet, daß die Anweisungsbewertung bis zur nächsten Anweisung verzögert werden muß "Klammer zu" **')'**. Zum Beispiel wird die folgende Sequenz :

```
AND(        IX12
OR          IX35
)
```

so interpretiert :

Ergebnis := Ergebnis AND (IX12 OR IX35)

B.8.2. AWL- Operatoren

Die folgende Tabelle ist eine Zusammenfassung sämtlicher Operatoren der AWL-Sprache:

Operator	*Ergänzung*	*Operand*	*Beschreibung*
LD	N	Variable,Konst	Lädt den Operand
ST	N	Variable	Speichert aktuelles Ergebnis
S		BOO-Variable	Forcierung zu TRUE
R		BOO-Variable	Forcierung zu FALSE
AND (&)	N (	BOO	logisches UND
OR	N (	BOO	logisches ODER
XOR	N (	BOO	exklusiv-ODER
ADD	(	ANA oder TMR	Addition

SUB	(	ANA oder TMR	Substraktion
MUL	(	ANA oder TMR	Multiplikation
DIV	(	ANA oder TMR	Division
GT	(	ANA oder TMR	Test >
GE	(	ANA oder TMR	Test >=
EQ	(	ANA oder TMR	Test =
LE	(	ANA oder TMR	Test <=
LT	(	ANA oder TMR	Test <
NE	(	ANA oder TMR	Test <>
JMP	C N	Marke	Sprung zu einer Marke
RET	C N		Rückgabe von Unterprogramm
CAL	C N	Funktionsbaustein	
		Instancen-Name	Aufruf eines Funktionsbausteins
)			Ausführung verzögerter Anweisungen

Im folgenden Abschnitt werden ausschließlich Operatoren beschrieben, die spezifisch zur AWL-Sprache sind. Andere Standard-Operatoren werden im Abschnitt "Standard-Operatoren, Funktionsbausteine und Funktionen" erläutert.

B.8.2.1. "LD"-Operator

Operation	Lädt einen Wert in das aktuelle Ergebnis
Ergänzungen	N
Operand	Konstanter Ausdruck Interne, Ein- oder Ausgangsvariable

Beispiel :

```
                                    (* BEISPIELE VON LD-OPERATIONEN *)
LDex:       LD      false           (* Ergebnis := boolesche Konstante FALSE *)
            LD      true            (* Ergebnis := boolesche Konstante TRUE *)
            LD      123             (* Ergebnis := analoge Konstante *)
            LD      123.4           (* Ergebnis := reale analoge Konstante *)
            LD      t#3s            (* Ergebnis := Timer-Konstante *)
            LD      boo_var1        (* Ergebnis := boolesche Variable *)
            LD      ana_var1        (* Ergebnis := analoge Variable *)
            LD      tmr_var1        (* Ergebnis := Timer-Variable *)
            LDN     boo_var2        (* Ergebnis := NOT ( boolesche Variable ) *)
```

B.8.2.2. Operator "ST"

Operation	Speichert das aktuelle Ergebnis in einer Variablen Das aktuelle Ergebnis wird von dieser Anweisung nicht geändert
Ergänzungen	N
Operand	Interne oder Ausgangsvariable

Beispiel :

```
                                    (* BEISPIELE VON ST-OPERATIONEN *)
```

```
STboo:      LD      false
            ST      boo_var1    (* boo_var1 := FALSE *)
            STN     boo_var2    (* boo_var2 := TRUE *)
STana:      LD      123
            ST      ana_var1    (* ana_var1 := 123 *)
STtmr:      LD      t#12s
            ST      tmr_var1    (* tmr_var1 := t#12s *)
```

B.8.2.3. "S"-Operator

Operation Forciert eine boolesche Variable zu TRUE, wenn das aktuelle Ergebnis TRUE ist. Keine Operation erfolgt, wenn das aktuelle Ergebnis FALSE ist. Das aktuelle Ergebnis wird von dieser Anweisung nicht geändert.
Ergänzungen (keine)
Operand Boolesche, interne oder Ausgangsvariable

Beispiel :

```
                                (* BEISPIELE VON S-OPERATIONEN*)
SETex:      LD      true        (* aktuelles Ergebnis := TRUE *)
            S       boo_var1    (* boo_var1 := TRUE *)
                                (* unverändertes, aktuelles Ergebnis *)
            LD      false       (* aktuelles Ergebnis := FALSE *)
            S       boo_var1    (* keine Aktion - boo_var1 unverändert *)
```

B.8.2.4. "R"-Operator

Operation Forciert eine boolesche Variable zu FALSE, wenn das aktuelle Ergebnis TRUE ist. Keine Operation erfolgt, wenn das aktuelle Ergebnis FALSE ist. Das aktuelle Ergebnis wird von dieser Anweisung nicht geändert.
Ergänzungen (keine)
Operand Boolesche, interne oder Ausgangsvariable

Beispiel :

```
                                (* BEISPIELE VON R-OPERATIONEN *)
RESETex:    LD      true        (* aktuelles Ergebnis := TRUE *)
            R       boo_var1    (* boo_var1 := FALSE *)
                                (* aktuelles Ergebnis unverändert *)
            ST      boo_var2    (* boo_var2 := TRUE *)
            LD      false       (* aktuelles Ergebnis := FALSE *)
            R       boo_var1    (* keine Aktion - boo_var1 unverändert *)
```

B.8.2.5. "JMP"-Operator

Operation Sprung zu einer durch eine Marke aufgefundene Anweisung
Ergänzungen C N
Operand Marke in demselben AWL-Programm definiert

Beispiel :

```
(* Das folgende Beispiel testet den Wert eines analogen Selektors *)
(* (0 oder 1 oder 2) und forciert eine von drei booleschen Variablen. *)
(* Der Test "gleich 0" wird von der JMPC-Anweisung ausgeführt. *)

JMPex:    LD      selector   (* Selektor = 0 oder 1 oder 2 *)
          BOO                (* Umrechnung zu boolesch *)
          JMPC    test1      (* wenn Selector = 0 dann *)
          LD      true
          ST      bo0        (* bo0 := true *)
          JMP     JMPend     (* Programmende *)
                             (* dekrementiert Selektor *)
test1:    LD      selector
          SUB     1          (* Selektor ist 0 oder 1 *)
          BOO                (* Umrechnung zu boolesch *)
          JMPC    test2      (* wenn Selektor = 0 dann *)
          LD      true
          ST      bo1        (* bo1 := true *)
          JMP     JMPend     (* Programmende *)
                             (* letzte Möglichkeit *)
test2:    LD      true
          ST      bo2        (* bo2 := true *)
JMPend:                      (* Programmende *)
```

B.8.2.6. "RET"-Operator

Operation	Beendet die Programmausführung. Wenn die AWL-Sequenz ein Unterprogramm ist, wird das aktuelle Ergebnis des Werts zum aufrufenden Programm zurückgegeben.
Ergänzungen	C N
Operand	(keine)

Beispiel :

```
(* Das folgende Beispiel testet den Wert eines analogen Selektors *)
(* (0 oder 1 oder 2) und forciert eine von drei booleschen Variablen. *)
(* Der Test "gleich 0" wird von der JMPC-Anweisung ausgeführt. *)

RETex:    LD      selector   (* Selektor = 0 oder 1 oder 2 *)
          BOO                (* Umrechnung zu boolesch *)
          JMPC    test1      (* wenn Selektor = 0 dann *)
          LD      true
          ST      bo0        (* bo0 := true *)
          RET                (* Ende - gibt 0 zurück *)
                             (* dekrementiert Selektor *)
test1:    LD      selector
          SUB     1          (* Selektor = 0 oder 1 *)
          BOO                (* Umrechnung zu boolesch *)
```

```
            JMPC    test2       (* wenn Selektor = 0 dann *)
            LD      true
            ST      bo1         (* bo1 := true *)
            LD      1           (* Wert des Selektors *)
            RET                 (* Ende - gibt 1zurück *)
                                (* letzte Möglichkeit *)
test2:      RETNC               (* Rückgabe, wenn Selektor einen *)
                                (* ungültigen Wert enthält *)
            LD      true
            ST      bo2         (* bo2 := true *)
            LD      2           (* Wert des Selektors *)
            RET                 (* Ende - gibt 2 zurück *)
```

B.8.2.7. ")"-Operator

Operation Führt eine verzögerte Operation aus.
Die verzögerte Operation wurde durch "(" geändert.
Ergänzungen (keine)
Operand (keine)

Beispiel :

```
(* Beispiel von verschachtelten, verzögerten Operationen *)
(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)
Delayed:    LD      a1          (* Ergebnis:= a1; *)
            ADD(    a2          (* verzögertes ADD - Ergebnis:= a2; *)
            MUL(    a3          (* verzögertes MUL - Ergebnis:= a3; *)
            SUB     a4          (* Ergebnis:= a3 - a4; *)
            )                   (* führt verz. MUL aus - Ergebnis:= a2*(a3-a4); *)
            MUL     a5          (* Ergebnis:= a2 * (a3 - a4) * a5; *)
            )                   (* führt verzögertes ADD aus*)
                                (* Ergebnis:= a1 + (a2 * (a3 - a4) * a5); *)
            ADD     a6          (* Ergebnis:= a1 + (a2 * (a3 - a4) * a5) + a6; *)
            ST      res         (* res := aktuelles Ergebnis *)
```

B.8.2.8. Aufruf eines Unterprogramms oder einer Funktion

Ein Unterprogramm oder eine Funktion (in der AWL, ST, KOP, FBS oder "C"-Sprache geschrieben) wird von der AWL-Sprache aufgerufen, wobei ihr Name als Operator benutzt wird.

Operation führt ein Unterprogramm oder eine Funktion aus - der vom Unterprogramm oder der Funktion zurückgegebene Wert wird im aktuellen AWL-Ergebnis gespeichert.

Ergänzungen (keine)
Operand Der erste Aufrufparameter muß vor dem Aufruf im aktuellen Ergebnis gespeichert werden. Die folgenden Aufrufparameter werden, durch Kommas getrennt, im Operandenfeld ausgedrückt.

Beispiel :

```
(* Aufruf eines Unterprogramms : Umrechnung analog > Timer *)
Main:       LD      bi0
            SUBPRO  bi1,bi2     (* Unterprogramm-Aufruf *)
                                (* Ergebnis := vom Unterprogramm *)
                                (* zurückgegebener Wert *)
            ST      result      (* Ergebnis := vom Unterprogramm zurückgegebener
Wert *)
            GT      vmax        (* Überschreitungstest *)
            RETC                (* Rückgabe, wenn Überschreitung *)
            LD      result
            MUL     1000        (* Umrechnung in Millisekunden *)
            ST      tmval       (* Speichern in einer Timer-Variablen *)

 (* Unterprogramm mit Namen 'SUBPRO' : bewertet den analogen Wert *)
(* als binärer Wert auf drei booleschen Eingängen gegeben: in0, in1, in2 sind die drei
booleschen Eingangsparameter des Unterprogramms *)
            LD      in2
            ANA                 (* Ergebnis:= ana(in2); *)
            MUL     2           (* Ergebnis:= 2*ana(in2); *)
            ST      temporary   (* temporary := result *)
            LD      in1
            ANA
            ADD     temporary   (* Ergebnis:= 2*ana(in2) + ana(in1); *)
            MUL     2           (* Ergebnis:= 4*ana(bi2) + 2*ana(bi1); *)
            ST      temporary   (* temporary := result *)
            LD      in0
            ANA
            ADD     temporary   (* Ergebnis:= 4*ana(in2) + 2*ana(in1)+ana(in0); *)
            ST      SUBPRO      (* gibt den Wert des aktuellen Ergebnisses zurück*)
                                (* an das aufrufende Programm *)
```

B.8.2.9. Aufruf eines Funktionsbausteins: CAL-Operator

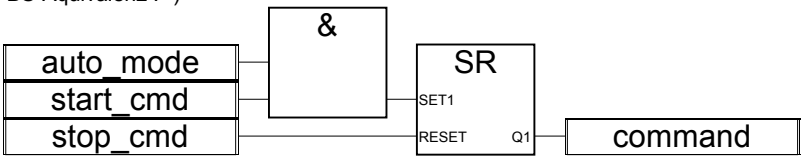
Operation	ruft einen Funktionsbaustein auf
Ergänzungen	C N
Operand	Name der Funktionsbaustein-Instance. Vor dem Aufruf, der die LD/ST-Sequenz benutzt, müssen die Eingangsparameter des Bausteins zugewiesen werden. Wenn Ausgangsparameter benutzt werden, sind diese bekannt.

Beispiel 1:

```
(* Aufruf des Funktionsbausteins SR : SR1 ist eine Instance von SR *)
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
```

```
LD      SR1.Q1
ST      command
```

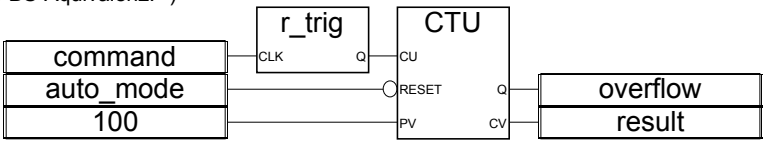
(* FBS-Äquivalenz : *)



Beispiel 2
(*Es wird angenommen, daß R_TRIG1 eine Instance des Bausteins R_TRIG ist und CTU1 eine Instance des Bausteins CTU *)

```
LD      command
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
LDN     auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
CAL     CTU1
LD      CTU1.Q
ST      overflow
LD      CTU1.cv
ST      result
```

(* FBS-Äquivalenz: *)



B.9. Standard Operatoren, F-Bausteine und Funktionen

B.9.1. Standard-Operatoren

Es folgt eine Liste der verfügbaren Funktionsbausteine, die den in der **ST**-Sprache definierten Standard-Operatoren, -Funktionsbausteinen und -Funktionen entsprechen. Die folgende Typen-Abkürzungen wurden benutzt: BOO (boolesch), INT (ganzzahlig), REAL (real), TMR (Timer) und MSG (Zeichenkette) :

- Datenmanipulation ..... Zuweisung, analoge Negation
- Boolesche Operationen ..... Logisches UND
  Logisches ODER
  Logisches exklusiv-ODER
- Arithmetische Operationen ..... Addition
  Substraktion
  Multiplikation
  Division
- Maskierungs-Operationen ..... Analoge Bit-zu-Bit UND-Maske
  Analoge Bit-zu-Bit ODER-Maske
  Analoge Bit-zu-Bit exklusiv-ODER-Maske
  Bit-zu-Bit Inversion
- Vergleiche ..... Kleiner als
  Kleiner oder gleich
  Größer als
  Größer oder gleich
  Gleich
  Ungleich
- Datenumrechnungen ..... Umrechnung zu boolesch
  Umrechnung zu analog, ganzzahlig
  Umrechnung zu analog, real
  Umrechnung zu Timer
  Umrechnung zu Zeichenkette
- Weitere Operatoren ..... Verkettung von Zeichenketten
  Zugriff auf die Systemparameter
  E/A-Verarbeitung

1 gain

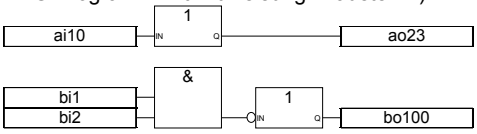
Argumente:

IN	alle Typen
Q	alle Typen

Beschreibung:
Zuweisung einer Variablen an eine andere Variable.

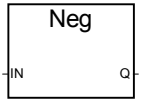
Dieser Baustein wird insbesondere dann benutzt, wenn ein Diagramm-Eingang direkt mit einem Diagramm-Ausgang verbunden werden soll. Er kann auch verwendet werden, um eine boolesche Inversion vor dem Anschluß einer Verbindungslinie an eine Diagramm-Ausgangsvariable einzufügen.

(* FBS-Programm mit "Zuweisung"-Baustein *)




```
(* ST-Äquivalenz: *)
ao23 := ai10;
bo100 := NOT (bi1 AND bi2);

(* AWL-Äquivalenz: *)
LD          ai10
ST          ao23
LD          bi1
AND         bi2
STN         bo100
```

NEG

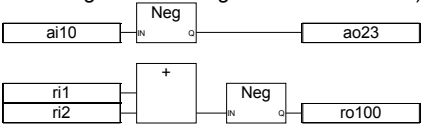


Argumente:

IN	INT- REAL	Ein- und Ausgang müssen das gleiche Format haben.
Q	INT - REAL	

Beschreibung:
Zuweisung der Negation einer Variablen.

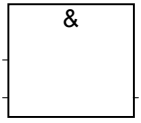
(*FBS-Programm mit Negations-Bausteinen *)




```
(* ST-Äquivalenz *)
ao23 := - (ai10);
```

```
ro100 := - (ri1 + ri2);

(* AWL-Äquivalenz: *)
LD          ai10
MUL         -1
ST          ao23
LD          ri1
ADD         ri2
MUL         -1.0
ST          ro100
```

& AND



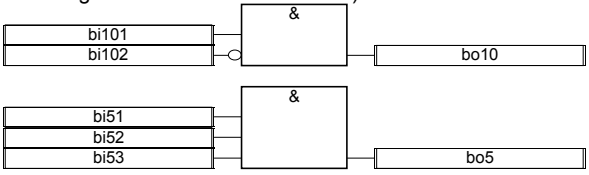
Anmerkung: Für diesen Operator kann die Anzahl der Eingänge zu mehr als zwei erweitert werden.

Argumente :

(inputs)	BOO	
output	BOO	Logisches UND zwischen beiden Termen

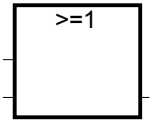
Beschreibung:
Boolesches UND zwischen zwei oder mehr Termen.

(* FBS-Programm mit "UND"-Baustein *)




```
(* ST-Äquivalenz: *)
bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;
(* AWL-Äquivalenz: *)
            LD          bi101       (*Ergebnis := bi101 *)
            ANDN        bi102       (*Ergebnis := bi101 AND not(bi102) *)
            ST          bo10        (* bo10 := Ergebnis  *)
            LD          bi51        (*Ergebnis := bi51;
            &           bi52        (*Ergebnis := bi51 AND bi52 *)
            &           bi53        (*Ergebnis := (bi51 AND bi52) AND bi53 *)
            ST          bo5         (* bo5 := Ergebnis  *)
```

>=1 OR



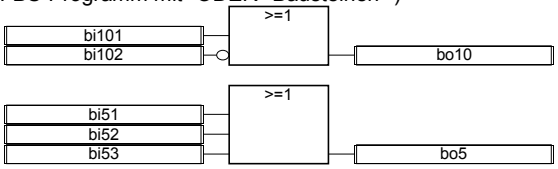
Anmerkung: Für diesen Operator kann die Anzahl der Eingänge zu mehr als zwei erweitert werden.

Argumente :

(inputs)	BOO	
output	BOO	Logisches ODER zwischen den beiden Termen

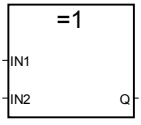
Beschreibung:
Boolesches ODER von zwei oder mehr Termen.

(* FBS-Programm mit "ODER"-Bausteinen *)




```
(* ST-Äquivalenz: *)
bo10 := bi101 OR NOT (bi102);
bo5 := (bi51 OR bi52) OR bi53;

(* AWL-Äquivalenz: *)
LD      bi101
ORN     bi102
ST      bo10
LD      bi51
OR      bi52
OR      bi53
ST      bo5
```

=1 XOR



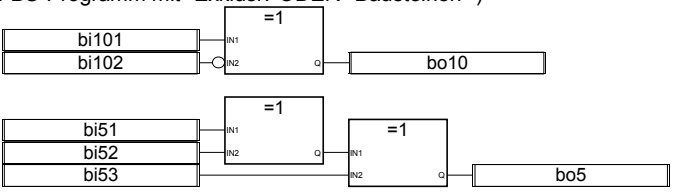
Argumente:

IN1	BOO
IN2	BOO

Q	BOO	boolesches exklusiv-ODER der 2 Eingangstermen

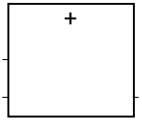
Beschreibung:
Boolesches exklusiv-ODER zwischen zwei Termen.

(* FBS-Programm mit "Exklusiv ODER"-Bausteinen *)




```
(* ST-Äquivalenz: *)
bo10 := bi101 XOR NOT (bi102);
bo5 := (bi51 XOR bi52) XOR bi53;

(* AWL-Äquivalenz: *)
LD      bi101
XORN    bi102
ST      bo10
LD      bi51
XOR     bi52
XOR     bi53
ST      bo5
```

+



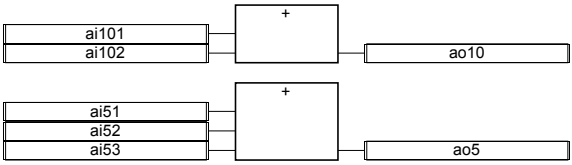
Anmerkung: Für diesen Operator kann die Anzahl der Eingänge zu mehr als zwei erweitert werden.

Argumente:

(inputs)	INT-REAL	kann GANZZAHLIG oder REAL sein (alle Argumente müssen das gleiche Format haben)
output	INT -REAL	vorzeichenbehaftete Addition der beiden Termen

Beschreibung:
Addition von zwei oder mehr analogen Variablen.

(* FBS-Programm mit "Addition"-Baustein *)




```
(* ST-Äquivalenz: *)
ao10 := ai101 + ai102;
ao5 := (ai51 + ai52) + ai53;

(* AWL-Äquivalenz: *)
LD          ai101
ADD         ai102
ST          ao10
LD          ai51
ADD         ai52
ADD         ai53
ST          ao5
```

-

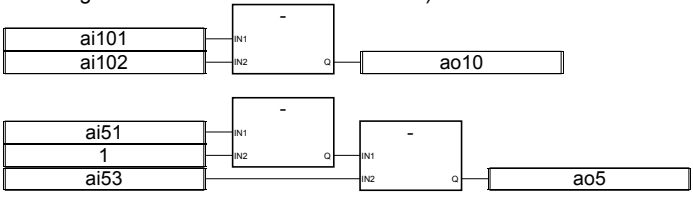
Argumente:

IN1	INT -REAL	kann GANZZAHLIG oder REAL sein
IN2	INT -REAL	(IN1 und IN2 müssen das gleiche Format haben)
Q	INT -REAL	Substraktion (erste - zweite)

Beschreibung:

Substraktion von zwei analogen Variablen (erste - zweite).

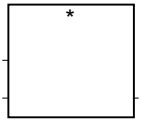
(* FBS-Programm mit "Substraktion"-Baustein *)




```
(* ST-Äquivalenz: *)
ao10 := ai101 - ai102;
ao5 := (ai51 - 1) - ai53;
```

```
(* AWL-Äquivalenz: *)
LD          ai101
SUB         ai102
ST          ao10
LD          ai51
SUB         1
SUB         ai53
ST          ao5
```

*



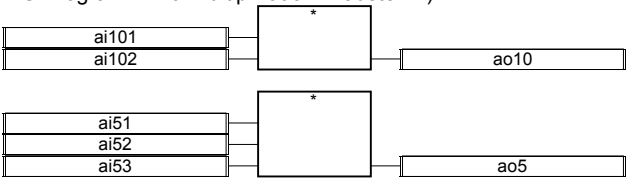
Anmerkung: Für diesen Operator kann die Anzahl der Eingänge zu mehr als zwei erweitert werden.

Argumente:

(inputs)	INT -REAL	kann GANZZAHLIG oder REAL sein (alle Argumente müssen das gleiche Format haben)
output	INT -REAL	vorzeichenbehaftete Multiplikation der Eingangstermen

Beschreibung:
Multiplikation von zwei oder mehr analogen Variablen.

(* FBS-Programm mit "Multiplikation"-Baustein *)




```
(* ST-Äquivalenz: *)
ao10 := ai101 * ai102;
ao5 := (ai51 * ai52) * ai53;
```

```
(* ST-Äquivalenz: *)
LD          ai101
MUL         ai102
ST          ao10
LD          ai51
MUL         ai52
MUL         ai53
ST          ao5
```

/

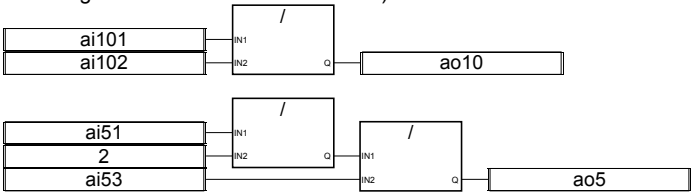
Argumente:

IN1	INT -REAL	kann GANZZAHLIG oder REAL sein
IN2	INT-REAL	analoger Wert ungleich null (Divisor) (IN1 und IN2 müssen das gleiche Format haben)
Q	INT-REAL	vorzeichenbehaftete ganzzahlige oder reale Division von IN1 durch IN2

Beschreibung:

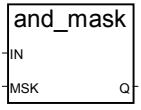
Division von zwei analogen Variablen (die erste wird durch die zweite geteilt).

(* FBS-Programm mit "Division"-Baustein *)




```
(* ST-Äquivalenz: *)
ao10 := ai101 / ai102;
ao5 := (ai51 / 2) / ai53;

(* AWL-Äquivalenz: *)
LD      ai101
DIV     ai102
ST      ao10
LD      ai51
DIV     2
DIV     ai53
ST      ao5
```

AND_MASK



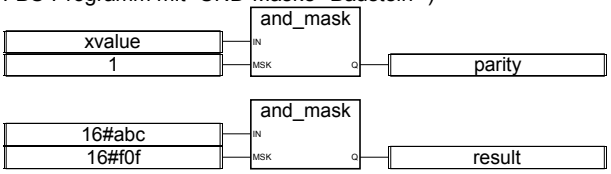
Argumente:

IN	INT	muß vom Format GANZZAHLIG sein
MSK	INT	muß vom Format GANZZAHLIG sein

Q	INT	Bit-zu-Bit logisches UND zwischen IN und MSK

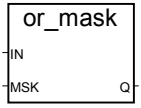
Beschreibung:
Ganzzahlige analoge Bit-zu-Bit UND-Maske.

(* FBS-Programm mit "UND-Maske"-Baustein *)




```
(* ST-Äquivalenz: *)
parity := AND_MASK (xvalue, 1); (* =1, wenn xvalue ungerade *)
result := AND_MASK (16#abc, 16#f0f); (* gleich 16#a0c *)

(* AWL-Äquivalenz: *)
LD          xvalue
AND_MASK    1
ST          parity
LD          16#abc
AND_MASK    16#f0f
ST          result
```

OR_MASK

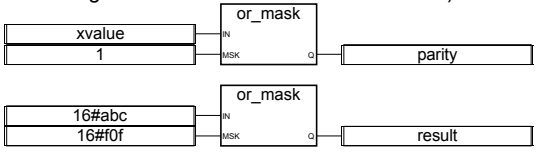


Argumente:

IN	INT	muß vom Format GANZZAHLIG sein
MSK	INT	muß vom Format GANZZAHLIG sein
Q	INT	logisches Bit-zu-Bit ODER zwischen IN und MSK

Beschreibung:
Ganzzahlige analoge Bit-zu-Bit ODER-Maske.

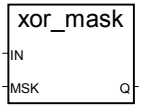
(* FBS-Programm mit "ODER-Maske"-Baustein *)



(* ST-Äquivalenz: *)

```
is_odd := OR_MASK (xvalue, 1); (* immer ungerade *)
result := OR_MASK (16#abc, 16#f0f); (* gleich 16#fbf *)

(* AWL-Äquivalenz: *)
LD          xvalue
OR_MASK     1
ST          is_odd
LD          16#abc
OR_MASK     16#f0f
ST          result
```

XOR_MASK



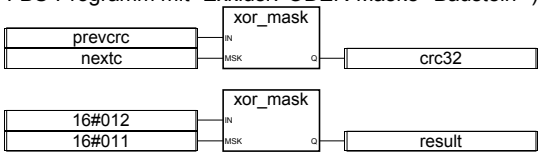
Argumente:

IN	INT	muß vom Format GANZZAHLIG sein
MSK	INT	muß vom Format GANZZAHLIG sein
Q	INT	Logisches Bit-zu-Bit exklusiv-ODER zwischen IN und MSK

Beschreibung:

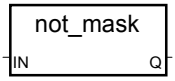
Ganzzahlige analoge Bit-zu-Bit exklusiv-ODER-Maske

(* FBS-Programm mit "Exklusiv ODER-Maske"-Baustein *)




```
(* ST-Äquivalenz: *)
crc32 := XOR_MASK (prevcrc, nextc);
result := XOR_MASK (16#012, 16#011); (*gleich 16#003 *)

(*AWL-Äquivalenz: *)
LD          prevcrc
XOR_MASK    nextc
ST          crc32
LD          16#012
XOR_MASK    16#011
ST          result
```

NOT_MASK

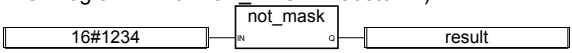


Argumente :

IN	INT	muß vom Format GANZZAHLIG sein
Q	INT	invertiert Bit zu Bit den IN auf 32 Bits

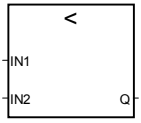
Beschreibung:
Ganzzahlige analoge Bit-zu-Bit Negation-Maske

(*FBS-Programm mit "NOT_MASK"-Baustein *)




```
(* ST-Äquivalenz: *)
result := NOT_MASK (16#1234);
(* result ist 16#FFFF_EDCB *)

(* AWL-Äquivalenz: *)
LD          16#1234
NOT_MASK
ST          result
```

<

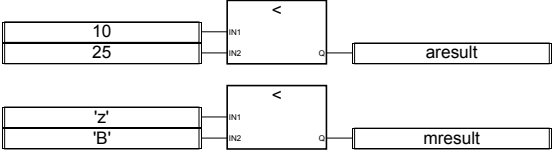


Argumente:

IN1	INT-REAL-TMR-MSG	
IN2	INT-REAL-TMR-MSG	die beiden Eingänge müssen vom gleichen Typ sein
Q	BOO	TRUE wenn IN1 < IN2

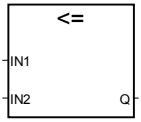
Beschreibung:
Testet, ob eine Wert KLEINER ALS ein anderer Wert ist (Analog, Timer oder Zeichenkette)

(* FBS-Programm mit "Kleiner als"-Baustein *)



```
(* ST-Äquivalenz: *)
aresult := (10 < 25);     (* aresult = TRUE *)
mresult := ('z' < 'B');    (* mresult = FALSE *)

(* AWL-Äquivalenz: *)
LD          10
LT          25
ST          aresult
LD          'z'
LT          'B'
ST          mresult
```

<=



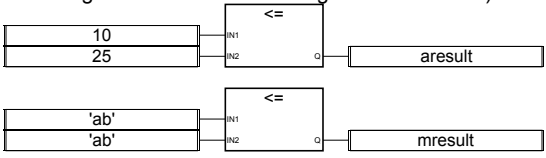
Argumente:

IN1	INT-REAL-MSG	
IN2	INT-REAL-MSG	die beiden Eingänge müssen vom gleichen Typ sein
Q	BOO	TRUE wenn IN1 <= IN2

Beschreibung:

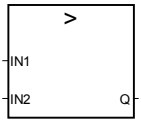
Testet, ob ein Wert KLEINER als ODER GLEICH einem anderen Wert ist (Analog oder Zeichenkette)

(* FBS-Programm mit "Kleiner oder gleich"-Baustein *)




```
(* ST-Äquivalenz: *)
aresult := (10 <= 25);     (* aresult = TRUE *)
mresult := ('ab' <= 'ab');    (* mresult = TRUE *)

(* AWL-Äquivalenz: *)
LD          10
LE          25
ST          aresult
LD          'ab'
LE          'ab'
ST          mresult
```

>



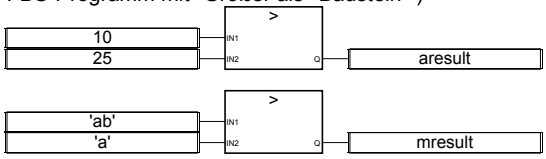
Argumente:

IN1	INT-REAL-TMR-MSG	
IN2	INT-REAL-TMR-MSG	die beiden Eingänge müssen vom gleichen Typ sein
Q	BOO	TRUE wenn IN1 > IN2

Beschreibung:

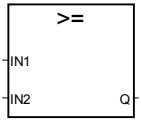
Testet, ob ein Wert GRÖSSER ALS ein anderer Wert ist (Analog, Timer oder Zeichenkette)

(* FBS-Programm mit "Größer als"-Baustein *)




```
(* ST-Äquivalenz: *)
aresult := (10 > 25);     (* aresult = FALSE *)
mresult := ('ab' > 'a');   (* mresult = TRUE *)

(* AWL-Äquivalenz: *)
LD      10
GT      25
ST      aresult
LD      'ab'
GT      'a'
ST      mresult
```

>=

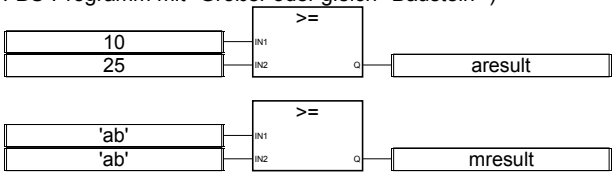


Argumente:

IN1	INT-REAL-MSG	
IN2	INT-REAL-MSG	die beiden Eingänge müssen vom gleichen Typ sein
Q	BOO	TRUE wenn IN1 >= IN2

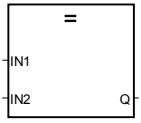
Beschreibung:
Testet, ob ein Wert GRÖSSER als oder GLEICH einem anderen Wert ist (Analog oder Zeichenkette)

(* FBS-Programm mit "Größer oder gleich"-Baustein *)




```
(* ST-Äquivalenz: *)
aresult := (10 >= 25);      (* aresult = FALSE *)
mresult := ('ab' >= 'ab'); (* mresult = TRUE *)

(* AWL-Äquivalenz: *)
LD          10
GE          25
ST          aresult
LD          'ab'
GE          'ab'
ST          mresult
```

=

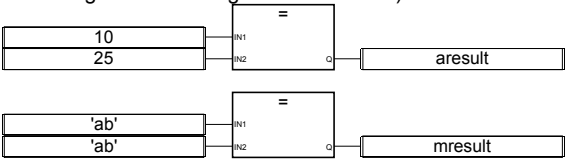


Arguments:

IN1	INT-REAL-MSG	
IN2	INT-REAL-MSG	die beiden Eingänge müssen vom gleichen Typ sein
Q	BOO	TRUE wenn IN1 = IN2

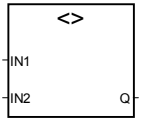
Beschreibung:
Testet, ob ein Wert GLEICH einem anderen Wert ist (Analog oder Zeichenkette)

(* FBS-Programm mit "Ist gleich"-Baustein *)



```
(* ST-Äquivalenz: *)
aresult := (10 = 25);     (* aresult = FALSE *)
mresult := ('ab' = 'ab');    (* mresult = TRUE *)

(* AWL-Äquivalenz: *)
LD          10
EQ          25
ST          aresult
LD          'ab'
EQ          'ab'
ST          mresult
```

<>



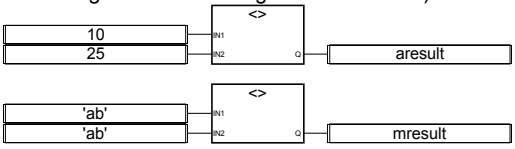
Arguments:

IN1	INT-REAL-MSG	
IN2	INT-REAL-MSG	die beiden Eingänge müssen vom gleichen Typ sein
Q	BOO	TRUE wenn erster <> zweiter

Beschreibung:

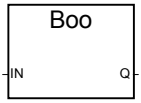
Testet, ob ein Wert UNGLEICH einem anderen Wert ist (Analog oder Zeichenkette)

(* FBS-Programm mit "Ist ungleich"-Baustein *)




```
(* ST-Äquivalenz: *)
aresult := (10 <> 25);       (* aresult = TRUE *)
mresult := ('ab' <> 'ab');  (* mresult = FALSE *)

(* AWL-Äquivalenz: *)
LD          10
NE          25
ST          aresult
LD          'ab'
NE          'ab'
ST          mresult
```

BOO



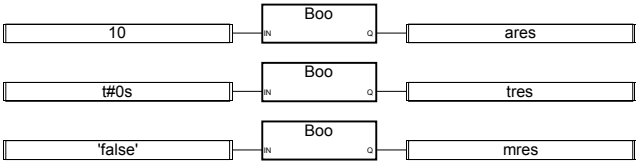
Argumente:

IN	ALLE TYPEN	alle nicht booleschen Werte
Q	BOO	TRUE für einen analogen Wert ungleich null FALSE für einen analogen Wert gleich null TRUE für eine Zeichenkette 'TRUE' FALSE für eine Zeichenkette 'FALSE'

Beschreibung:

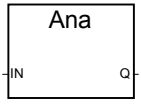
Beliebige Variable in eine boolesche Variable umrechnen.

(* FBS-Programm mit Umrechnung zu boolesch *)




```
(* ST-Äquivalenz: *)
ares := BOO (10);        (* ares = TRUE *)
tres := BOO (T#0s);      (* tres  FALSE *)
mres := BOO ('false');   (* mres = FALSE *)

(* AWL-Äquivalenz: *)
LD          10
BOO
ST          ares
LD          t#0s
BOO
ST          tres
LD          'false'
BOO
ST          mres
```

ANA



Argumente:

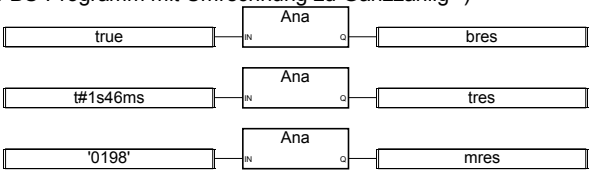
IN	ALLE TYPEN	alle Werte, die nicht analog ganzzahlig sind
Q	INT	0 wenn IN ist FALSE / 1 wenn IN ist TRUE

Zahl in Millisekunden für einen Timer
ganzzahliger Teil für reale analoge Werte

Dezimalwert, durch eine Zeichenkette dargestellt

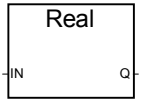
Beschreibung:
Beliebige Variable zu einer ganzzahligen Variablen umrechnen.

(* FBS-Programm mit Umrechnung zu Ganzzahlig *)




```
(* ST-Äquivalenz: *)
bres := ANA (true);              (* bres = 1 *)
tres := ANA (t#1s46ms);     (* tres = 1046 *)
mres := ANA ('0198');            (* mres = 198 *)

(* AWL-Äquivalenz: *)
LD          true
ANA
ST          bres
LD          t#1s46ms
ANA
ST          tres
LD          '0198'
ANA
ST          mres
```

REAL

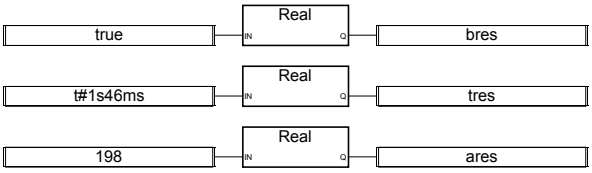


Argumente:

IN	BOO-INT-TMR	alle Werte, die nicht analog, real sind (keine Zeichenkette)
Q	REAL	0.0 wenn IN ist FALSE / 1.0 wenn IN ist TRUE Zahl in Millisekunden für einen Timer Zahl äquivalent zu einem analogen, ganzzahligen Wert

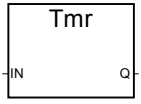
Beschreibung:
Beliebige Variable in eine reale Variable umrechnen

(* FBS-Programm mit Umrechnung zu Real *)




```
(* ST-Äquivalenz: *)
bres := REAL (true);              (* bres = 1.0 *)
tres := REAL (t#1s46ms);   (* tres = 146.0 *)
ares := REAL (198);               (* ares = 198.0 *)

(* AWL-Äquivalenz: *)
LD              true
REAL
ST              bres
LD              t#1s46ms
REAL
ST              tres
LD              198
REAL
ST              ares
```

TMR



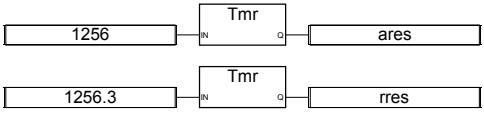
Argumente:

IN	INT-REAL	alle Werte außer Zeitwerten IN (oder ganzzahliger Teil von IN , wenn real) ist die Zahl in Millisekunden
Q	TIMER	Zeitwert, dargestellt durch IN

Beschreibung:

Beliebige Variable in eine Timer-Variable umrechnen.

(* FBS-Programm mit Umrechnung zu Timer *)




```
(* ST-Äquivalenz: *)
ares := TMR(1256);              (* ares = t#1s256ms *)
```

```
rres := TMR (1256.3);          (* rres = t#1s256ms *)

(* AWL-Äquivalenz: *)
LD          1256
TMR
ST          ares
LD          1256.3
TMR
ST          rres
```

MSG

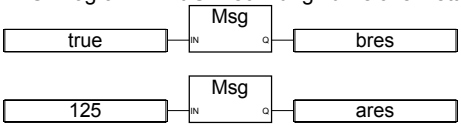
Argumente:

IN	BOO- INT-REA	alle Werte, außer Zeichenketten
Q	ZK	'false' oder 'true', wenn IN boolesch ist dezimale Darstellung, wenn IN analog ist

Beschreibung:

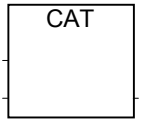
Beliebige Variable in eine Zeichenketten-Variable umrechnen.

(* FBS-Programm mit Umrechnung zu Zeichenkette *)




```
(* ST-Äquivalenz: *)
bres := MSG (true); (* bres = 'TRUE' *)
ares := MSG (125); (* ares = '125' *)

(* AWL-Äquivalenz: *)
LD          true
MSG
ST          bres
LD          125
MSG
ST          ares
```

CAT



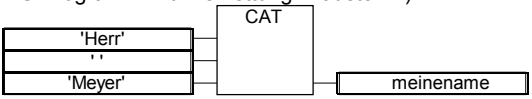
Anmerkung: Für diesen Operator kann die Anzahl der Eingänge zu mehr als zwei erweitert werden.

Argumente:

(inputs)	ZK	(die Summe der Längen der Zeichenketten darf die Kapazität des Ergebnisses nicht überschreiten)
output	ZK	Verkettung der Eingangs-Zeichenketten

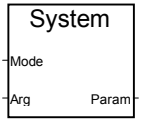
Beschreibung:
Mehrere Zeichenketten zu einer Zeichenkette verketten

(* FBS-Programm mit "Verkettung"-Baustein *)




```
(* ST-Äquivalenz *)
meinname:= ('Herr' + ' ') + 'Meyer';
(* bedeutet: meinname := 'Herr Meyer' *)

(* AWL-Äquivalenz *)
LD      'Herr'
ADD     ' '
ADD     'Meyer'
ST      meinname
```

SYSTEM



Argumente:

Mode	INT	identifiziert den Systemparameter und den Zugriff-Modus
Arg	INT -TMR	neuer Wert für einen "Schreib"-Zugriff
Param	INT	Wert des zugegriffenen Parameters

Beschreibung:
Zugriff auf die Systemparameter

Für die SYSTEM-Funktion sind folgende Befehle (vordefinierte Schlüsselwörter) verfügbar :

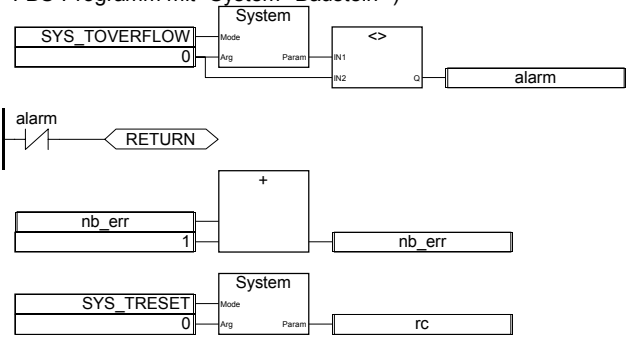
Befehl	*Bedeutung*
SYS_TALLOWED	Lesen der programmierten Zykluszeit
SYS_TCURRENT	Lesen der aktuellen Zykluszeit
SYS_TMAXIMUM	Lesen der maximalen Zykluszeit
SYS_TOVERFLOW	Lesen der Zykluszeit-Überschreitungen
SYS_TRESET	Einstellen des Zeitzählers auf null
SYS_TWRITE	Schreiben der Zykluszeit
SYS_ERR_TEST	Test der Ausführungsfehler
SYS_ERR_READ	Lesen des letzten Ausführungsfehlers

Hier die Liste der Argumente, die für die vordefinierten Befehle der SYSTEM-Funktion erforderlich sind :

Befehl	*Argument*
SYS_TALLOWED	0
SYS_TCURRENT	0
SYS_TMAXIMUM	0
SYS_TOVERFLOW	0
SYS_TRESET	0
SYS_TWRITE	neue zugelassene Zykluszeit
SYS_ERR_TEST	0
SYS_ERR_READ	0

Befehl	*zurückgegebener Wert*
SYS_TALLOWED	zugelassene Zykluszeit
SYS_TCURRENT	aktuelle Zykluszeit
SYS_TMAXIMUM	festgestellte max. Zykluszeit
SYS_TOVERFLOW	Anzahl der Überschreitungen
SYS_TRESET	0
SYS_TWRITE	geschriebener Wert
SYS_ERR_TEST	0, wenn keine Fehler gefunden wurden
SYS_ERR_READ	ältester Fehlercode

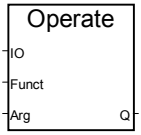
(* FBS-Programm mit "System"-Baustein *)




```
(* ST-Äquivalenz *)
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
If (alarm) Then
```

```
        nb_err := nb_err + 1;
        rc := SYSTEM (SYS_TRESET, 0);
End_If;
```

OPERATE



Argumente:

IO	(ALLE)	Ein- oder Ausgangsvariable
Funct	INT	auszuführende Aktion
Arg	INT	Argument für die ausgeführte Aktion
Q	INT	Rückgabe Ausführungsbericht

Beschreibung:
Zugriff auf einen E/A-Kanal

Die Bedeutung der Argumente von OPERATE hängen von der Implementierung der E/A-Schnittstelle ab. Siehe Handbuch der verwendeten Steuerung oder Datenblatt der entsprechnden E/A für weitere Informationen über die Funktion OPERATE.

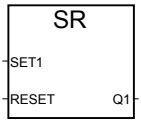
B.9.2. Standard-Funktionsbausteine

Die folgenden Standard-Funktionsbausteine werden vom ISaGRAF System unterstützt. Sie sind vordefiniert und brauchen nicht in der Bibliothek deklariert zu werden.

● Boolesche Daten	SR	Bistabil - Forcierung zu TRUE prioritär
	RS	Bistabil - Forcierung zu FALSE prioritär
	R_Trig	Erkennung einer steigenden Flanke
	F_Trig	Erkennung einer fallenden Flanke
	SEMA	Semaphor
● Zählen	CTU	Zähler
	CTD	Zwischenzähler
	CTUD	Zähler/ Zwischenzähler
● Timer	TON	Timer bei Einschaltung
	TOF	Timer bei Ausschaltung
	TP	Impuls-Timer
● Ganzzahlige	CMP	Komparator
	STACKINT	Stapel analoger, ganzzahliger Werte
● Reale	AVERAGE	Aktueller Mittelwert auf N Samples
	HYSTER	Bool. Hysterese ü.d. Differenz
	LIM_ALRM	Grenzalarm mit Hystereseberechnung
	INTEGRAL	Integration (Zeit)
	DERIVATE	Abweichung (Zeit)
● Signale	BLINK	Blinklicht
	SIG_GEN	Zeitbasis

Anmerkung: Neu entwickelte "C"-Funktionsbausteine werden für die FBS-Sprache verfügbar.

SR



Argumente:

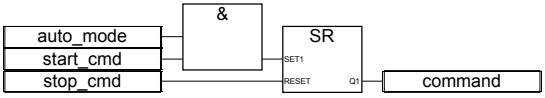
SET1	BOO	wenn TRUE, forciert Q1 zu TRUE (dominant)
RESET	BOO	wenn TRUE, setzt Q1 auf FALSE zurück
Q1	BOO	boolescher Speicherstatus

Beschreibung:

Dominant bistabil forcieren: Siehe folgende True-Tabelle:

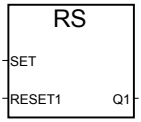
Set1	Reset	Q1	Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(* FBS-Programm mit "SR"-Funktionsbaustein *)




```
(* ST-Äquivalenz: Es wird angenommen, daß SR1 eine Instance des Bausteins SR ist *)
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;

(* AWL-Äquivalenz: *)
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
```

RS



Argumente:

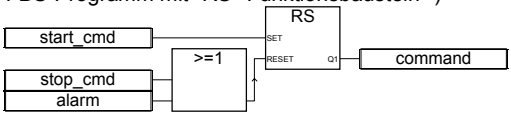
SET	BOO	wenn TRUE, forciert Q1 zu TRUE
RESET1	BOO	wenn TRUE, setzt Q1 auf FALSE zurück (dominant)
Q1	BOO	boolescher Speicherstatus

Beschreibung:

Dominant bistabil rücksetzen: Siehe folgende True-Tabelle:

Set	Reset1	Q1	Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(* FBS-Programm mit "RS"-Funktionsbaustein *)



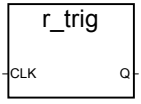
(* ST-Äquivalenz: Es wird angenommen, daß RS1 eine Instance des Bausteins RS ist *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```

(* AWL-Äquivalenz: *)

```
LD      start_cmd
ST      RS1.set
LD      stop_cmd
OR      alarm
ST      RS1.reset1
CAL     RS1
LD      RS1.Q1
ST      command
```

R_TRIG

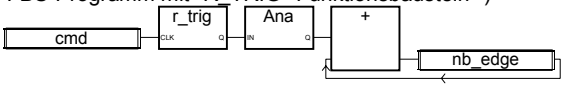


Argumente:

CLK	BOO	alle booleschen Variablen
Q	BOO	TRUE wenn CLK von FALSE zu TRUE übergeht FALSE in allen anderen Fällen

Beschreibung:
Erkennung einer steigenden Flanke einer booleschen Variablen

(* FBS-Programm mit "R_TRIG"-Funktionsbaustein *)



(* ST-Äquivalenz: Es wird angenommen, daß R_TRIG1 eine Instance des Bausteins R_TRIG ist *)
R_TRIG1(cmd);
nb_edge := ANA(R_TRIG1.Q) + nb_edge;

(* AWL-Äquivalenz: *)
LD cmd
ST R_TRIG1.clk
CAL R_TRIG1
LD R_TRIG1.Q
ANA
ADD nb_edge
ST nb_edge

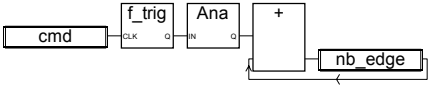
F_TRIG

Argumente:

CLK	BOO	alle booleschen Variablen
Q	BOO	TRUE wenn CLK von TRUE auf FALSE übergeht FALSE in allen anderen Fällen

Beschreibung:
Erkennung einer fallenden Flanke einer booleschen Variablen

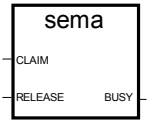
(* FBS-Programm mit "F_TRIG"-Funktionsbaustein *)



(* ST-Äquivalenz: Es wird angenommen, daß F_TRIG1 eine Instance des Bausteins F_TRIG ist *)
F_TRIG1(cmd);
nb_edge := ANA(F_TRIG1.Q) + nb_edge;

```
(* AWL-Äquivalenz: *)
LD          cmd
ST          F_TRIG1.clk
CAL         F_TRIG1
LD          F_TRIG1.Q
ANA
ADD         nb_edge
ST          nb_edge
```

SEMA



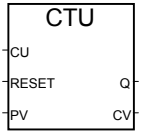
Argumente:

CLAIM	BOO	"test and set" Befehl
RELEASE	BOO	Freigabe des Semaphor
BUSY	BOO	Status des Semaphor

Beschreibung:

```
(* "x" ist eine boolesche Variable zu FALSE initialisiert*)
busy := x;
If claim Then
        x := True;
Else
        If release Then
                busy := False;
                x := False;
        End_if;
End_if;
```

CTU



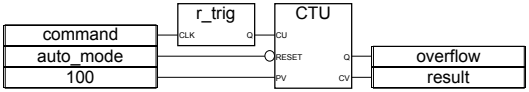
Argumente:

CU	BOO	Zähleingang (zählt, wenn CU = TRUE)
RESET	BOO	Zurücksetzen auf null (prioritär)
PV	INT	programmierter Maximalwert
Q	BOO	overflow: TRUE wenn CV = PV
CV	INT	Aktuelles Zählergebnis

Achtung: Der CTU-Baustein erkennt nicht die steigenden oder fallenden Flanken des Zähleingangs (CU). Er muß einem "R_TRIG" oder "F_TRIG"-Baustein zugeordnet werden, wenn ein Impuls-Zähler erstellt werden soll.

Beschreibung:
Schrittweise zählen (ganzzahlig) von 0 aufwärts bis zu einem gegebenen Wert.

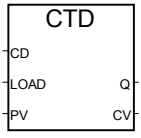
(* FBS-Programm mit "CTU"-Funktionsbaustein *)




```
(* ST-Äquivalenz: Es wird angenommen, daß R_TRIG1 eine Instance des Bausteins R_TRIG
ist und CTU1 eine Instance des Bausteins CTU *)
CTU1(R_TRIG1(command),NOT(auto_mode),100);
overflow := CTU1.Q;
result := CTU1.CV;
```

```
(* AWL-Äquivalenz: *)
LD      command
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
LDN     auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
CAL     CTU1
LD      CTU1.Q
ST      overflow
LD      CTU1.cv
ST      result
```

CTD



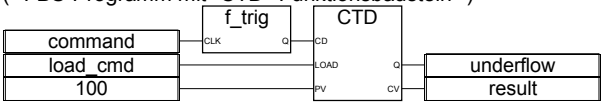
Argumente:

CD	BOO	Zähleingang (zählt abwärts, wenn CD = TRUE)
LD	BOO	Ladebefehl (prioritär) (CV = PV, wenn LD = TRUE)
PV	INT	Programmierter Initialwert
Q	BOO	underflow: TRUE, wenn CV = 0
CV	INT	Aktuelles Zählergebnis

Achtung: Der CTD-Baustein erkennt nicht die steigenden oder fallenden Flanken des Zähleingangs (CU). Er muß einem "R_TRIG" oder "F_TRIG"-Baustein zugeordnet werden, wenn ein Impuls-Zähler erstellt werden soll.

Beschreibung:
Schrittweise zählen (ganzzahlig) von einem gegebenen Wert abwärts bis 0

(* FBS-Programm mit "CTD"-Funktionsbaustein *)



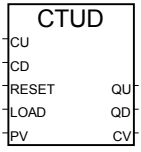
(* ST-Äquivalenz: Es wird angenommen, daß F_TRIG1eine Instance des Bausteins F_TRIG ist und CTD1 eine Instance des Bausteins CTD *)

```
CTD1(F_TRIG1(command),load_cmd,100);
underflow := CTD1.Q;
result := CTD1.CV;
```

(* AWL-Äquivalenz: *)

```
LD      command
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ST      CTD1.cd
LD      load_cmd
ST      CTD1.load
LD      100
ST      CTD1.pv
CAL     CTD1
LD      CTD1.Q
ST      underflow
LD      CTD1.cv
ST      result
```

CTUD



Argumente:

CU	BOO	Zählen (Inkrementation, wenn CU = TRUE)
CD	BOO	Zwischenzählen (Dekrementation, wenn CD=TRUE)
R	BOO	zurück-auf-null-Befehl (prioritär) (CV = 0, wenn RESET = TRUE)
LD	BOO	Ladebefehl (CV = PV, wenn LOAD = TRUE)
PV	INT	Programmierter Maximalwert

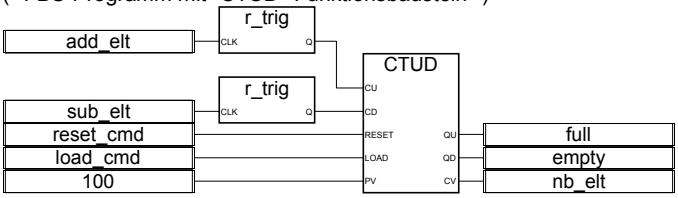
QU	BOO	Signal "voll": TRUE, wenn CV = PV
QD	BOO	Signal "leer": TRUE, wenn CV = 0
CV	INT	Aktuelles Zählergebnis

Achtung: Der CTUD-Baustein erkennt nicht die steigenden oder fallenden Flanken des Zähleingangs (CU). Er muß einem "R_TRIG" oder "F_TRIG"-Baustein zugeordnet werden, wenn ein Impuls-Zähler erstellt werden soll.

Beschreibung:

Schrittweise zählen (ganzzahlig) von 0 aufwärts bis zu einem gegebenen Wert oder von einem gegebenen Wert abwärts bis 0

(* FBS-Programm mit "CTUD"-Funktionsbaustein *)

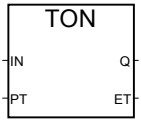


(* ST-Äquivalenz: Es wird angenommen, daß R_TRIG1 und R_TRIG2 zwei Instancen des Bausteins R_TRIG sind und daß CTUD1 eine Instance des Bausteins CTUD ist*)CTUD1(R_TRIG1(add_elt), R_TRIG2(sub_elt), reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;

```
(* AWL-Äquivalenz: *)
LD      add_elt
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTUD1.cu
LD      sub_elt
ST      R_TRIG2.clk
CAL     R_TRIG2
LD      R_TRIG2.Q
ST      CTUD1.cd
LD      reset_cmd
ST      CTUD1.reset
LD      load_cmd
ST      CTUD1.load
LD      100
ST      CTUD1.pv
CAL     CTUD1
LD      CTUD1.QU
ST      full
LD      CTUD1.QD
ST      empty
LD      CTUD1.CV
```

ST nb_elt

TON



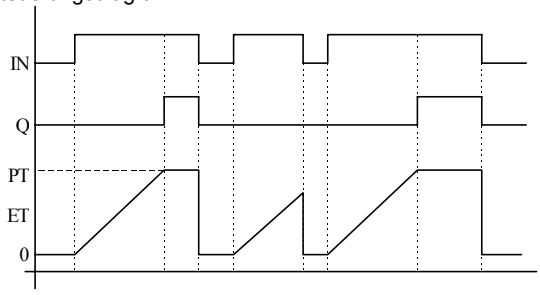
Argumente:

IN	BOO	Wenn steigende Flanke, started Inkrementierung des internen Timers Wenn fallende Flanke, stoppt internen Timer und setzt ihn zurück
PT	TMR	Programmierte Maximalzeit
Q	BOO	Wenn TRUE, ist die programmierte Zeit abgelaufen
ET	TMR	aktuelle verstrichene Zeit

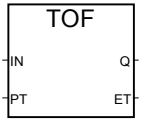
Beschreibung:

Einen internen Timer bis zu einem bestimmten Wert inkrementieren.

Steuerungsdiagramm:



TOF

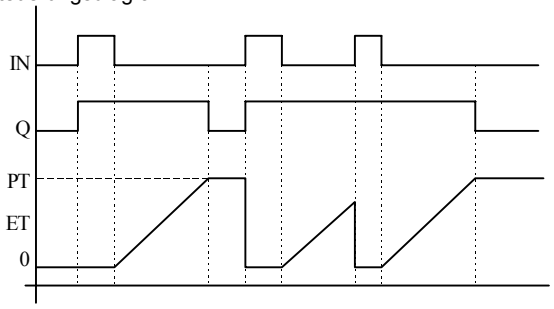


Argumente:

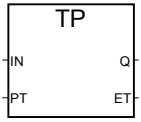
IN	BOO	Wenn fallende Flanke, startet Inkrementierung des internen Timers Wenn steigende Flanke, stoppt internen Timer und setzt ihn zurück
PT	TMR	Programmierte Maximalzeit
Q	BOO	Wenn TRUE: Gesamtzeit nicht abgelaufen
ET	TMR	aktuelle verstrichene Zeit

Beschreibung:
Einen internen Timer bis zu einem bestimmten Wert inkrementieren.

Steuerungsdiagramm:



TP

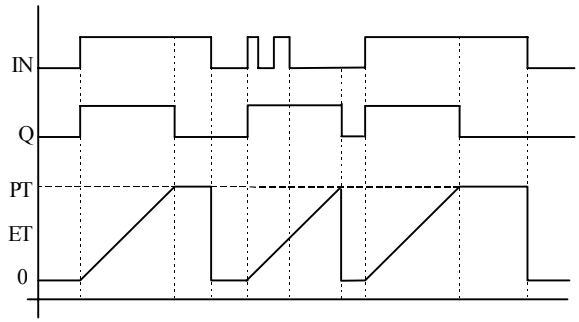


Argumente:

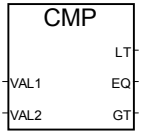
IN	BOO	Wenn steigende Flanke, startet Inkrementierung des internen Timers (wenn noch nicht inkrementiert) Wenn FALSE und nur wenn Timer abgelaufen ist, setzt den internen Timer zurück Änderungen auf IN während des Zählens sind wirkungslos
PT	TMR	Programmierte Maximalzeit
Q	BOO	Wenn TRUE: Timer zählt
ET	TMR	aktuelle verstrichene Zeit

Beschreibung:
Einen internen Timer bis zu einem bestimmten Wert inkrementieren.

Steuerungsdiagramm:



CMP



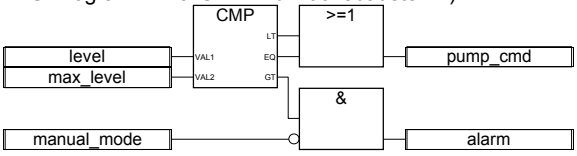
Argumente:

VAL1	INT	alle vorzeichenbehafteten, analogen, ganzzahligen Werte
VAL2	INT	alle vorzeichenbehafteten, analogen, ganzzahligen Werte
LT	BOOL	TRUE, wenn val1 kleiner als val2 ist
EQ	BOOL	TRUE, wenn val1 gleich val2 ist
GT	BOOL	TRUE, wenn val1 größer als val2 ist

Beschreibung:

Vergleich von zwei Werten: Gibt Auskunft, ob die Werte gleich sind oder der erste Werte kleiner oder größer als der zweite Wert ist.

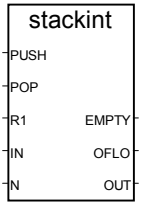
(* FBS-Programm mit "CMP"-Funktionsbaustein *)




```
(* ST-Äquivalenz: Es wird angenommen, daß CMP1eine Instance des Bausteins CMP ist *)
CMP1(level, max_level);
pump_cmd := CMP1.LT OR CMP1.EQ;
alarm := CMP1.GT AND NOT(manual_mode);

(* AWL-Äquivalenz: *)
LD      level
ST      CMP1.val1
LD      max_level
```

```
ST      CMP1.val2
CAL     CMP1
LD      CMP1.LT
OR      CMP1.EQ
ST      pump_cmd
LD      CMP1.GT
ANDN    manual_mode
ST      alarm
```

STACKINT



Argumente:

PUSH	BOO	Befehl "einkellern" (auf steigender Flanke) Wert IN oben auf Stapel addieren
POP	BOO	Befehl "auskellern" (auf steigender Flanke) im Stapel den letzten gepuschten Wert löschen (oben auf dem Stapel)
R1	BOO	Rücksetzen des Stapels auf Leerstand
IN	INT	aufzustapelnder Wert
N	INT	von der Anwendung definierte Größe des Stapels
EMPTY	BOO	TRUE, wenn der Stapel leer ist
OFLO	BOO	overflow: TRUE, wenn der Stapel voll ist
OUT	INT	Oberster Stackwert

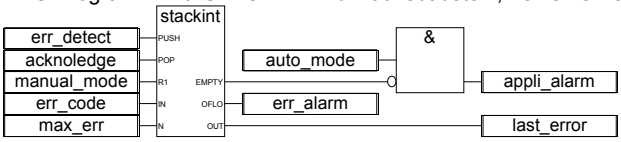
Beschreibung:

Einen Stapel (Stack) von ganzzahligen Werten verwalten.

Der Funktionsbaustein "STACKINT" beinhaltet die Erkennung von steigenden Flanken für die Eingänge PUSH und POP (Befehle "einkellern" und "auskellern"). Der Stapel kann im Höchstfall **128** Elemente enthalten. Die von der Anwendung bestimmte Größe darf nicht kleiner als 1 und nicht größer als 128 sein.

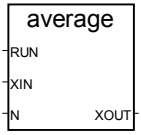
Beachten Sie, daß der Wert OFLO nur nach einem Rücksetzen gültig ist (R1 wurde mindestens einmal zu TRUE forciert und auf FALSE zurückgesetzt).

(* FBS-Programm mit "STACKINT"-Funktionsbaustein; Fehlerverwaltung *)



```
(* ST-Äquivalenz: Es wird angenommen, daß STACKINT1 eine Instance des Bausteins
STACKINT ist *)
STACKINT1(err_detect, acknoledge, manual_mode, err_code, max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;

(* AWL-Äquivalenz: *)
LD          err_detect
ST          STACKINT1.push
LD          acknoledge
ST          STACKINT1.pop
LD          manual_mode
ST          STACKINT1.r1
LD          err_code
ST          STACKINT1.IN
LD          max_err
ST          STACKINT1.N
CAL         STACKINT1
LD          auto_mode
ANDN        STACKINT1.empty
ST          appli_alarm
LD          STACKINT1.OFLO
ST          err_alarm
LD          STACKINT1.OUT
ST          last_error
```

AVERAGE



Argumente:

RUN	BOOL	TRUE=start / FALSE=erneute Initialisierung
XIN	REAL	alle analogen, realen Werte
N	BOOL	Anzahl der Samples (von der Anwendung definiert)
XOUT	REAL	aktueller Mittelwert des Eingangs XIN

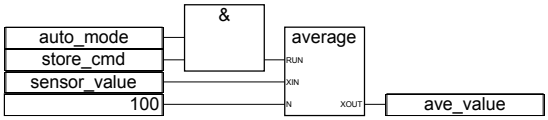
Beschreibung:

Speichert einen Wert bei jedem Zyklus und berechnet den Mittelwert sämtlicher bereits gespeicherten Werte. Nur die letzten N Werte werden gespeichert.

Die Anzahl der Samples darf **128** nicht überschreiten. Wenn der Eingang "**RUN**" den Wert **FALSE** hat (erneute Initialisierung), ist der Ausgangswert gleich dem Eingangswert.

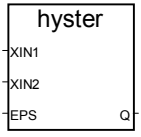
Wenn das Maximum N der gespeicherten Werte erreicht ist, wird der erste gespeicherte Wert vom letzten gespeicherten Wert überschrieben.

```
(* FBS-Programm mit "AVERAGE"-Funktionsbaustein : *)
```




```
(* ST-Äquivalenz: AVERAGE1 ist eine Instance des Bausteins AVERAGE *)
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value := AVERAGE1.XOUT;

(* AWL-Äquivalenz: *)
LD      auto_mode
AND     store_cmd
ST      AVERAGE1.run
LD      sensor_value
ST      AVERAGE1.xin
LD      100
ST      AVERAGE1.N
CAL     AVERAGE1
LD      AVERAGE1.XOUT
ST      ave_value
```

HYSTER



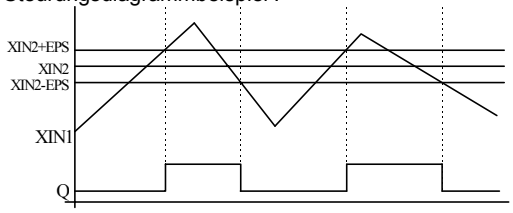
Argumente:

XIN1	REAL	alle analogen, realen Werte
XIN2	REAL	testet, ob XIN1den Wert XIN2+EPS überschritten hat
EPS	REAL	Wert der Hysterese (muß positif sein)
Q	BOO	TRUE, wenn XIN1 den Wert XIN2+EPS überschritten hat und noch nicht unter XIN2-EPS liegt

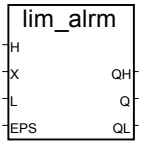
Beschreibung:

Hysterese auf einem realen Wert für einen Höchstwert

Steurungsdiagrammbeispiel :



LIM_ALRM



Argumente:

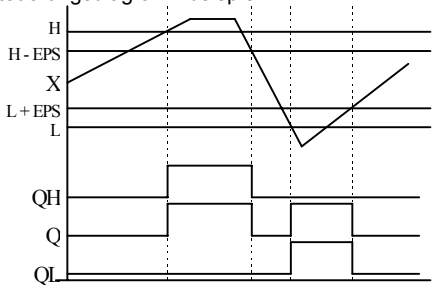
H	REAL	oberster Grenzwert
X	REAL	Eingang: alle analogen, realen Werte
L	REAL	unterster Grenzwert
EPS	REAL	Wert der Hysterese (muß positif sein)
QH	BOO	Alarm oberste Grenze: TRUE, wenn X über Höchstwert H liegt
Q	BOO	Alarm: Ausgang: TRUE, wenn X die Grenze überschritten hat
QL	BOO	Alarm unterste Grenze: TRUE, wenn X unter dem Tiefstwert L liegt

Beschreibung:

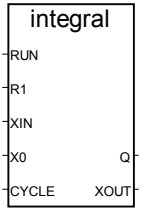
Hysterese auf einem realen Wert für Höchst- und Tiefstwerte.

Auf die oberste und unterste Grenze wird eine Hysterese angewendet. Das Delta der Hysterese, das für die oberste und unterste Grenze benutzt wird, ist die Hälfte des EPS-Parameterwerts.

Steuerungsdiagrammbeispiel :



INTEGRAL



Argumente:

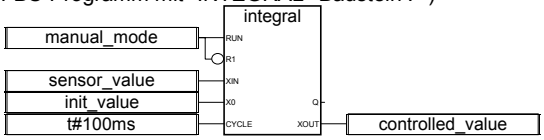
RUN	BOO	Modus: TRUE=Integration / FALSE=Warten
R1	BOO	zurück-auf-null-Befehl
XIN	REAL	Eingang: alle analoge, realen Werte
X0	REAL	Initialwert
CYCLE	TMR	Samples-Zeitspanne
Q	BOO	Not R1
XOUT	REAL	integrierter Wert

Beschreibung:

Integration eines realen Werts.

Wenn der Wert des Parameters "**CYCLE**" kleiner als die Zykluszeit der ISaGRAF Anwendung ist, dann ist die Sample-Zeitspanne gleich der Zykluszeit der Anwendung.

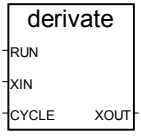
(* FBS-Programm mit "INTEGRAL"-Baustein : *)




```
(* ST-Äquivalenz: INTEGRAL1 Instance des Bausteins INTEGRAL *)
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value, t#100ms);
controlled_value := INTEGRAL1.XOUT;
```

```
(* AWL-Äquivalenz: *)
LD      manual_mode
ST      INTEGRAL1.run
STN     INTEGRAL1.R1
LD      sensor_value
ST      INTEGRAL1.XIN
LD      init_value
ST      INTEGRAL1.X0
LD      t#100ms
ST      INTEGRAL1.CYCLE
CAL     INTEGRAL1
LD      INTEGRAL1.XOUT
ST      controlled_value
```

DERIVATE



Argumente:

RUN	BOO	Modus: TRUE=normal / FALSE=zurück auf null

XIN	REAL	Eingang: alle analogen, realen Werte
CYCLE	TMR	Sample-Zeitspanne
XOUT	REAL	Ausgang: Derivat des Signals

Beschreibung:
Differentiation eines realen Werts.

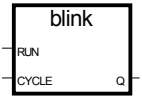
Wenn der Wert des Parameters "**CYCLE**" kleiner als die Zykluszeit der ISaGRAF Anwendung ist, dann ist die Sample-Zeitspanne gleich der Zykluszeit der Anwendung.

(* FBS-Programm mit "DERIVATE"-Baustein : *)


```
(* ST-Äquivalenz: DERIVATE1 ist eine Instance des Bausteins DERIVATE *)
DERIVATE1(manual_mode, sensor_value, t#100ms);
derivated_value := DERIVATE1.XOUT;
```

```
(* AWL-Äquivalenz: *)
LD          manual_mode
ST          DERIVATE1.run
LD          sensor_value
ST          DERIVATE1.XIN
LD          t#100ms
ST          DERIVATE1.CYCLE
CAL         DERIVATE1
LD          DERIVATE1.XOUT
ST      derivated_value
```

BLINK

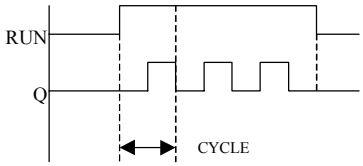


Argumente:

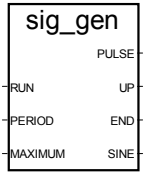
RUN	BOO	Modus: TRUE=blinken / FALSE=zurück auf null
CYCLE	TMR	Blinkperiode
Q	BOO	Ausgangs-Blinksignal

Beschreibung:
Generiert ein Blinksignal.

Steuerungsdiagramm :



SIG_GEN

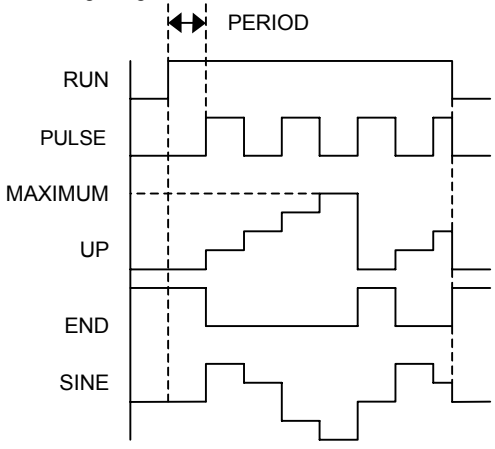


Argumente:

RUN	BOO	Modus: TRUE=run / FALSE=zurück auf null
PERIOD	TMR	Periode (Dauer) einer Probe
MAXIMUM	INT	Maximaler Zählwert
PULSE	BOO	Bei jeder Probe invertiert
UP	INT	Zähler bei jeder Probe inkrementiert
END	BOO	TRUE, wenn Zählinkrementierung beendet ist
SINUS	REAL	Sinus (Periode = Zählzeit)

Wenn der Zählvorgang den Höchstwert erreicht hat, beginnt er wieder bei 0 (null). Folglich behält END den Wert TRUE nur für 1 PERIOD.

Steuerungsdiagramm:



B.9.3. Standard-Funktionen

Die folgenden Standard-Funktionen werden vom ISaGRAF System unterstützt. Sie sind vordefiniert und brauchen in der Bibliothek nicht deklariert zu werden.

● Mathematische Funktionen....	ABS	Absolutwert
	EXPT, POW	Exponent, Potenzierung
	LOG	Logarithmus
	SQRT	Quadratwurzel
	TRUNC	Ganzer Teil
● Trigonometrische Funktionen	ACOS,ASIN,	Arcuscosinus, Arcussinus
	ATAN	Arcustangens,
	COS, SIN	Cosinus, Sinus
	TAN	Tangens
● Manipulation von Verzeichnissen	ROL, ROR	Rotation nach links,Rotation nach rechts
	SHL, SHR	Verschiebung nach links,Verschiebung nach rechts
● Daten-Manipulation	MIN, MAX	Minimum, Maximum
	LIMIT	Limit
	MOD	Modulo
	MUX4, MUX8	Multiplexer (4 oder 8 Eingänge)
	SEL	Binärer Selektor
	ODD	Ungerade Parität
	RAND	Zufallswert
● Daten-Umrechnung	ASCII	Zeichen → Ascii-Code
	CHAR	Ascii-Code → Zeichen
● Zeichenketten-Verwaltung	MLEN	Aktuelle Länge,
	DELETE	Zeichen Löschen
	INSERT	Zeichen Einsetzen,
	FIND	Zeichen Suchen,
	REPLACE	Zeichen Ersetzen,
	LEFT, MID	Extraktion links, in der Mitte,
	RIGHT	Extraktion rechts,
	DAY_TIME	Aktuelle(s) Datum, Zeit
● Tabellen-Verwaltung	ARCREATE	Tabelle erstellen
	ARREAD	Element Lesen,
	ARWRITE	Schreiben
● Binärdatei-Verwaltung	F_ROPEN	Öffnen einer Binärdatei im Lese-Modus
	F_WOPEN	Öffnen einer Binärdatei im Schreib-Modus
	F_CLOSE	Schließen einer Binärdatei
	F_EOF	Ende einer Binärdatei testen
	FA_READ	Einen analogen Wert in einer Binärdatei lesen
	FA_WRITE	Einen analogen Wert in einer Binärdatei schreiben
	FM_READ	Eine Zeichenkette in einer Binärdatei lesen
	FM_WRITE	Eine Zeichenkette in einer Binärdatei schreiben

Anmerkung: Wenn eine neue "C"-Funktion erstellt ist, wird sie in der Form eines Funktionsbausteins für die FBS-Sprache verfügbar.

ABS

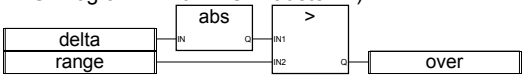
Argumente:

IN	REAL	alle vorzeichenbehafteten, realen Werte
Q	REAL	absoluter Wert (immer positiv)

Beschreibung:

Liefert den absoluten (positiven) Wert eines realen Werts.

(* FBS-Programm mit "ABS"-Baustein *)




```
(* ST-Äquivalenz *)
over := (ABS (delta) > range);

(* AWL-Äquivalenz: *)
LD      delta
ABS
GT      range
ST      over
```

EXPT

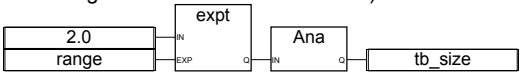
Argumente:

IN	REAL	alle vorzeichenbehafteten, realen Werte
EXP	INT	ganzer Exponent
Q	REAL	(IN ^{EXP})

Beschreibung:

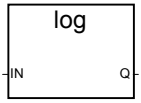
Liefert das Ergebnis der Operation: (Basis ^{Exponent}) 'Basis' ist das erste Argument und 'Exponent' das zweite.

(* FBS-Programm mit "EXPT"-Baustein *)



```
(* ST-Äquivalenz *)
tb_size := ANA (EXPT (2.0, range) );

(* AWL-Äquivalenz *)
LD          2.0
EXPT        range
ANA
ST          tb_size
```

LOG

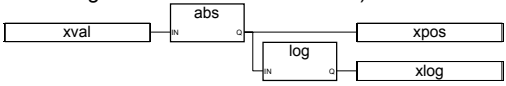


Argumente :

IN	REAL	muß größer als null sein
Q	REAL	Logarithmus (Basis 10) des Eingangswerts

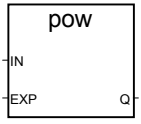
Beschreibung:
Berechnet den Logarithmus (Basis 10) eines realen Werts.

(* FBS-Programm mit "LOG"-Baustein *)




```
(* ST-Äquivalenz *)
xpos := ABS ( xval );
xlog := LOG ( xpos );

(* AWL-Äquivalenz *)
LD          xval
ABS
ST          xpos
LOG
ST          xlog
```

POW



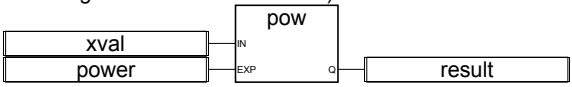
Argumente:

IN	REAL	zu potenzierende reale, analoge Zahl
EXP	REAL	Potenz von IN

Q	REAL	(IN ^{EXP}) 1.0 wenn IN ungleich 0.0 und EXP gleich 0.0 ist 0.0 wenn IN ist 0.0 und EXP negativ ist 0.0 wenn IN und EXP gleich 0.0 sind 0.0 wenn IN negativ und keine Ganzzahl ist

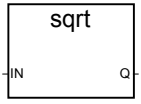
Beschreibung:
Liefert das reale Ergebnis der Operation: (Basis ^{Exponent}) 'Basis' ist das erste Argument und 'Exponent' das zweite. Der Exponent ist ein realer Wert.

(* FBS-Programm mit "POW"-Block *)




```
(* ST-Äquivalenz *)
result := POW (xval, power);

(* AWL-Äquivalenz *)
LD          xval
POW         power
ST          result
```

SQRT

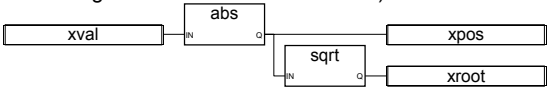


Argumente:

IN	REAL	muß größer oder gleich null sein
Q	REAL	Quadratwurzel des Eingangswerts

Beschreibung:
Berechnet die Quadratwurzel eines realen Werts.

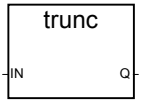
(* FBS-Programm mit "SQRT"-Baustein *)




```
(* ST-Äquivalenz *)
xpos := ABS ( xval );
xroot := SQRT (xpos);

(* AWL-Äquivalenz *)
LD          xval
ABS
```

```
ST      xpos
SQRT
ST      xrout
```

TRUNC



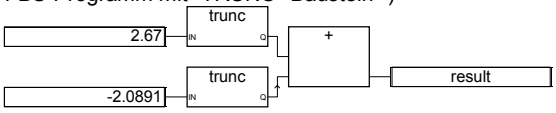
Argumente:

IN	REAL	alle vorzeichenbehafteten, realen Werte
Q	REAL	größter, ganzzahliger Wert; kleiner oder gleich Eingabewert, wenn IN > 0 kleinster, ganzzahliger Wert; kleiner oder gleich Eingabewert, wenn IN < 0

Beschreibung:

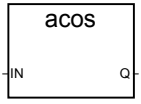
Rundet einen realen Wert zu einem ganzzahligen Wert auf

(* FBS-Programm mit "TRUNC"-Baustein *)




```
(* ST-Äquivalenz *)
result:= TRUNC (+2.67) + TRUNC (-2.0891);
(* bedeutet: result:= 2.0 + (-2.0) := 0.0 *)

(* AWL-Äquivalenz *)
LD      2.67
TRUNC
ST      temporary   (* temporäres Ergebnis des ersten TRUNC *)
LD      -2.0891
TRUNC
ADD     temporary
ST      result
```

ACOS



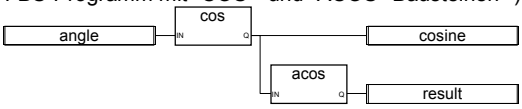
Argumente:

IN	REAL	muß sich im Intervall [-1.0 .. +1.0] befinden
Q	REAL	Arcuscosinus des Eingangswerts im Intervall [0.0 .. PI]

= 0.0 für einen ungültigen Eingang

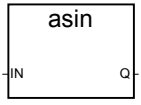
Beschreibung:
Berechnet den Arcuscosinus eines realen Werts.

(* FBS-Programm mit "COS"- und "ACOS"-Bausteinen *)




```
(* ST-Äquivalenz *)
cosine:= COS (angle);
result:= ACOS (cosine);   (* result ist gleich angle *)

(* AWL-Äquivalenz *)
LD          angle
COS
ST          cosine
ACOS
ST          result
```

ASIN

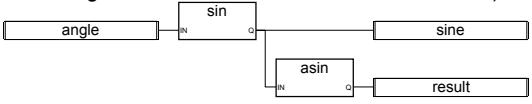


Argumente :

IN	REAL	muß sich im Intervall [-1.0 .. +1.0] befinden
Q	REAL	Arcussinus des Eingangswerts im Intervall [-PI/2 .. +PI/2] = 0.0 für einen ungültigen Eingang

Beschreibung:
Berechnet den Arcussinus eines realen Werts.

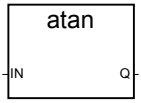
(* FBS-Programm mit "SIN"- und "ASIN"-Bausteinen *)




```
(* ST-Äquivalenz *)
Sine := SIN (angle);
result:= ASIN (sine);   (* result ist gleich angle *)

(* AWL-Äquivalenz *)
LD          angle
SIN
```

```
ST          sine
ASIN
ST          result
```

ATAN

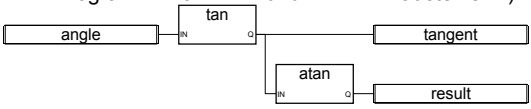


Argumente :

IN	REAL	alle analogen, realen Werte
Q	REAL	Arcustangens des Eingangswerts im Intervall [-PI/2 .. +PI/2] = 0.0 für einen ungültigen Eingang

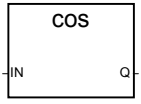
Beschreibung:
Berechnet den Arcustangens eines realen Werts.

(* FBD-Programm mit "TAN"- und "ATAN"-Bausteinen *)




```
(* ST-Äquivalenz *)
tangente := TAN (angle);
result:= ATAN (tangent);   (* result ist gleich angle *)

(* AWL-Äquivalenz *)
LD          angle
TAN
ST          tangent
ATAN
ST          result
```

COS

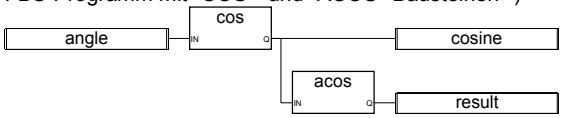


Argumente :

IN	REAL	alle analogen, realen Werte
Q	REAL	Cosinus - im Intervall [-1.0 .. +1.0]

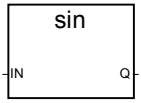
Beschreibung:
Berechnet den Cosinus eines realen Werts.

(* FBS-Programm mit "COS"- und "ACOS"-Bausteinen *)




```
(* ST-Äquivalenz *)
cosine := COS (angle);
result := ACOS (cosine); (* result ist gleich angle *)

(* AWL-Äquivalenz *)
LD          angle
COS
ST          cosine
ACOS
ST          result
```

SIN

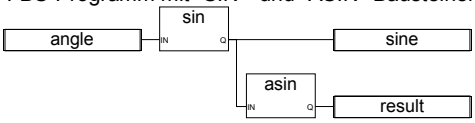


Argumente : **{**XE "SIN"**} {**XE "Sinus "**}**

IN	REAL	alle analogen, realen Werte
Q	REAL	Sinus des Eingangs - im Intervall [-1.0 .. +1.0]

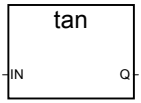
Beschreibung:
Berechnet den Sinus eines realen Werts.

(* FBS-Programm mit "SIN"- und "ASIN"-Bausteinen *)




```
(* ST-Äquivalenz *)
sine := SIN (angle);
result := ASIN (sine); (* result ist gleich angle *)

(* AWL-Äquivalenz *)
LD          angle
SIN
ST          sine
ASIN
ST          result
```

TAN

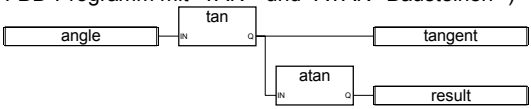


Argumente :

IN	REAL	kann nicht gleich PI/2 Modulo PI sein
Q	REAL	Tangens des Eingangswerts = 1E+38 für einen ungültigen Eingang

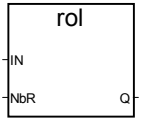
Beschreibung:
Berechnet den Tangens eines realen Werts.

(* FBD-Programm mit "TAN"- und "ATAN"-Bausteinen *)




```
(* ST-Äquivalenz *)
tangente := TAN (angle);
result:= ATAN (tangent);   (* result ist gleich angle *)

(* AWL-Äquivalenz *)
LD          angle
TAN
ST          tangent
ATAN
ST          result
```

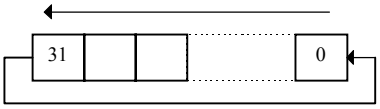
ROL



Argumente:

IN	**INT**	alle analogen, GANZZAHLIGEN Werte
NbR	**INT**	Anzahl der Rotationen von 1 Bit (im Intervall [1..31])
Q	**INT**	Wert nach links verschoben ohne Effekt, wenn NbR <= 0

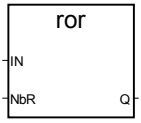
Beschreibung:
Bewirkt die Rotation der Bits einer Ganzzahl nach links. Die Rotation erfolgt auf 32 Bits:



(* FBS-Programm mit "ROL"-Baustein *)


```
(* ST-Äquivalenz *)
result := ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result   = 2#1001_1010_0110_1010*)

(* AWL-Äquivalenz *)
LD          register
ROL         1
ST          result
```

ROR

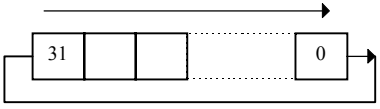


Argumente :

IN	INT	alle analogen, GANZZAHLIGEN Werte
NbR	INT	Anzahl der Rotationen von 1 Bit (im Intervall [1..31])
Q	INT	Wert nach rechts verschoben ohne Effekt, wenn NbR <= 0

Beschreibung:

Bewirkt die Rotation der Bits einer Ganzzahl nach rechts. Die Rotation erfolgt auf 32 Bits:

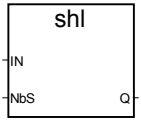


(* FBS-Programm mit "ROR"-Baustein *)


```
(* ST-Äquivalenz *)
result := ROR (register, 1);
(* register = 2#0100_1101_0011_0101 *)
```

```
(* result    = 2#1010_0110_1001_1010 *)

(* AWL-Äquivalenz *)
LD              register
ROR             1
ST              result
```

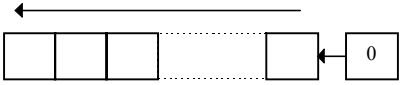
SHL



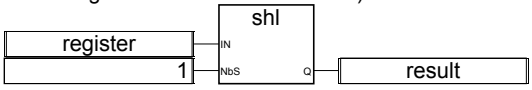
Argumente :

IN	INT	alle analogen, GANZZAHLIGEN Werte
NbS	INT	Anzahl der Verschiebungen von 1 Bit (im Intervall [1..31])
Q	INT	Wert nach links verschoben ohne Effekt, wenn NbS <= 0 der Wert 0 ersetzt das wertniedrigste Bit

Beschreibung:
Bewirkt die Verschiebung der Bits einer Ganzzahl nach links. Die Verschiebung erfolgt auf 32 Bits:

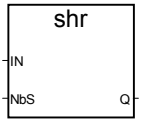


(* FBS-Programm mit "SHL"-Baustein *)




```
(* ST-Äquivalenz *)
result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result   = 2#1001_1010_0110_1010 *)

(* AWL-Äquivalenz *)
LD              register
SHL             1
ST              result
```

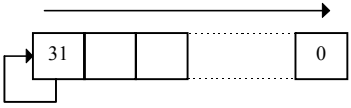
SHR



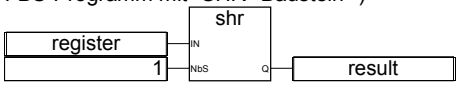
Argumente :

IN	INT	alle analogen, GANZZAHLIGEN Werte
NbS	INT	Anzahl der Verschiebungen von 1 Bit (im Intervall [1..31])
Q	INT	Wert nach rechts verschoben ohne Effekt, wenn NbS <= 0 das werthöchste Bit wird bei jeder Verschiebung kopiert

Beschreibung:
Bewirkt die Verschiebung der Bits einer Ganzzahl nach rechts. Die Verschiebung erfolgt auf 32 Bits:



(* FBS-Programm mit "SHR"-Baustein *)




```
(* ST-Äquivalenz *)
result := SHR (register,1);
(* register = 2#1100_1101_0011_0101 *)
(* result   = 2#1110_0110_1001_1010 *)

(* AWL-Äquivalenz *)
LD          register
SHR         1
ST          result
```

MIN

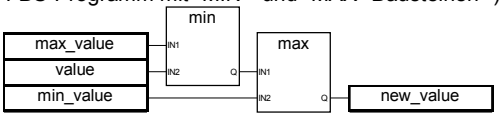
Argumente :

Wert_A	INT	alle vorzeichenbehafteten ganzzahligen Werte
Wert_B	INT	(kann nicht REAL sein)
Minimum	INT	Minimum der beiden Eingänge

Beschreibung:

Liefert das Minimum von zwei ganzzahligen Werten.

(* FBS-Programm mit "MIN"- und "MAX"-Bausteinen *)



(* ST-Äquivalenz *)
new_value := MAX (MIN (max_value, value), min_value);
(* bindet den Wert an die Forcierung [min_value..max_value] *)

(* AWL-Äquivalenz *)
LD max_value
MIN value
MAX min_value
ST new_value

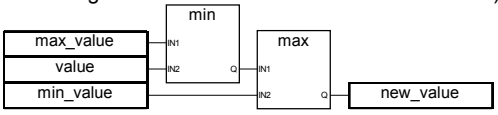
MAX

Argumente : **{**XE "MAX"**} {**XE "Maximum "**}**

IN1	INT	alle vorzeichenbehafteten, ganzzahligen Werte
IN2	INT	(kann nicht REAL sein)
Q	INT	Maximum der beiden Eingänge

Beschreibung:
Liefert das Maximum von zwei ganzzahligen Werten.

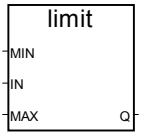
(* FBS-Programm mit "MIN"- und "MAX"-Bausteinen *)



(* ST-Äquivalenz *)
new_value := MAX (MIN (max_value, value), min_value);
(*bindet den Wert an die Forcierung [min_value..max_value] *)

(* AWL-Äquivalenz *)
LD max_value
MIN value
MAX min_value
ST new_value

LIMIT



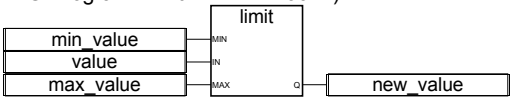
Argumente:

MIN	INT	Minimalwert
IN	INT	beliebiger analoger, ganzzahliger Wert
MAX	INT	Maximalwert
Q	INT	Grenzbereichwert

Beschreibung:

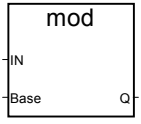
Begrenzt einen ganzzahligen Wert in ein bestimmtes Intervall. Ganz gleich, ob er seinen Wert beibehält, liegt dieser immer zwischen Minimum und Maximum, oder er wird zu Maximum forciert, wenn er darüber liegt, oder zu Minimum, wenn er darunter liegt.

(* FBS-Programm mit "LIMIT"-Block *)




```
(* ST-Äquivalenz: *)
new_value := LIMIT (min_value, value, max_value);
(* bindet den Wert an die Forcierung [min_value..max_value] *)

(* AWL-Äquivalenz: *)
LD          min_value
LIMIT       value, max_value
ST      new_value
```

MOD



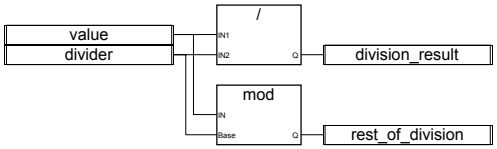
Argumente :

IN	INT	alle vorzeichenbehafteten, GANZZAHLIGEN Werte
Base	INT	muß größer als null sein
Q	INT	Modulo (Basis MOD Wert) ergibt -1, wenn Base <= 0

Beschreibung:

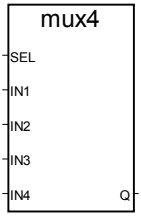
Berechnet das Modulo eines ganzzahligen Werts.

(* FBS-Programm mit "MOD"-Baustein *)




```
(* ST-Äquivalenz *)
division_result := (value / divider); ganze Division *)
rest_of_division := MOD (value, divider); Rest der Division *)

(* AWL-Äquivalenz *)
LD          value
DIV         divider
ST          division_result
LD          value
MOD         divider
ST      rest_of_division
```

MUX4



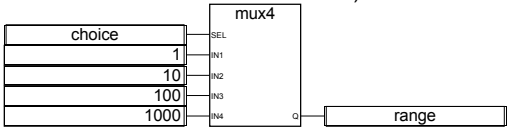
Argumente:

SEL	INT	ganzzahliger Wert im Intervall [0..3]
IN1..IN4	INT	vorzeichenbehaftete, ganzzahlige Werte
Q	INT	= IN1 wenn SEL = 0 = IN2 wenn SEL = 1 = IN3 wenn SEL = 2 = IN4 wenn SEL = 3 = 0 für alle anderen Fälle

Beschreibung:

Multiplexer mit 4 Eingängen: wählt einen Wert aus 4 ganzzahligen Werten.

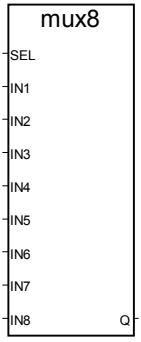
(* FBS-Baustein mit "MUX4"-Baustein *)



(* ST-Äquivalenz *)

```
range := MUX4 (choice, 1, 10, 100, 1000);
(* wählt aus 4 vordefinierten Bereichen, zum Beispiel, wenn choice 1 ist, ist range 10 *)

(* AWL-Äquivalenz *)
LD          choice
MUX4        1,10,100,1000
ST          range
```

MUX8



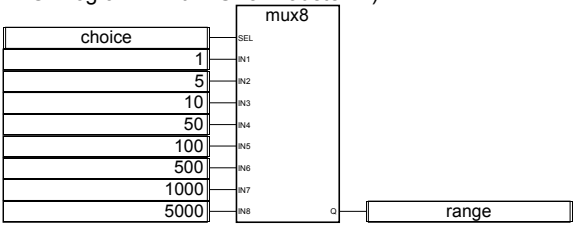
Argumente:

SEL	INT	ganzzahliger Wert im Intervall [0..7]
IN1..IN8	INT	vorzeichenbehaftete, ganzzahlige Werte
Q	INT	= IN1 wenn selector = 0 = IN2 wenn selector = 1 ... = IN8 wenn selector = 7 = 0 für alle anderen Fälle

Beschreibung:

Multiplexer mit 8 Eingängen: wählt einen Wert aus 8 ganzzahligen Werten.

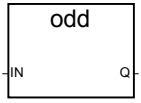
(* FBS-Programm mit "MUX8"-Baustein *)




```
(* ST-Äquivalenz *)
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);
(* wählt aus 8 vordefinierten Bereichen, zum Beispiel, wenn choice 3 ist, ist range 50 *)
```

```
(* AWL-Äquivalenz *)
LD          choice
MUX8        1,5,10,50,100,500,1000,5000
ST          range
```

ODD



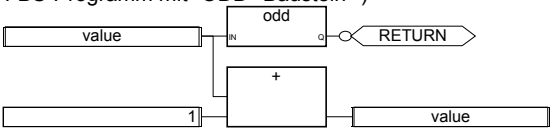
Argumente:

IN	INT	alle vorzeichenbehafteten, ganzzahligen Werte
Q	BOO	TRUE, wenn der Wert ungerade ist FALSE, wenn der Wert gerade ist

Beschreibung:

Testet die Parität einer Ganzzahl: Ergebnis gerade oder ungerade.

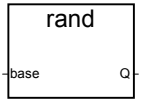
(* FBS-Programm mit "ODD"-Baustein *)




```
(* ST-Äquivalenz *)
If Not (ODD (value)) Then Return; End_if;
value := value + 1;
(* liefert immer gerade Werte *)
```

```
(* AWL-Äquivalenz *)
LD          value
ODD
RETNC
LD          value
ADD         1
ST          value
```

RAND

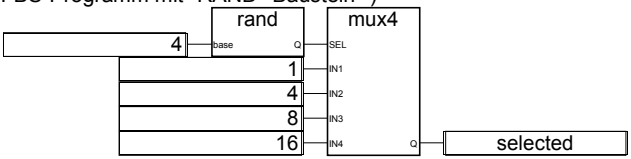


Argumene:

base	INT	definiert das Intervall der zugelassenen Werte
Q	INT	zufälliger Wert im Intervall [0..base-1]

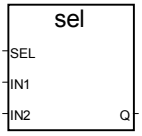
Beschreibung:
Liefert einen zufälligen ganzzahligen Wert in einem bestimmten Bereich.

(* FBS-Programm mit "RAND"-Baustein *)




```
(* ST-Äquivalenz *)
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
(*
Zufallswahl: 1 von 4 vordefinierten Werten
Der vom RAND-Aufruf gelieferte Wert ist im Satz [0..3],
also erhält 'selected', ausgegeben von MUX4, 'zufällig' den Wert
1, wenn 0 von RAND ausgegeben wird,
oder 4, wenn 1 von RAND ausgegeben wird,
oder 8, wenn 2 von RAND ausgegeben wird,
oder 16, wenn 3 von RAND ausgegeben wird
*)

(* AWL-Äquivalenz *)
LD          4
RAND
MUX4        1,4,8,16
ST          selected
```

SEL



Argumente:

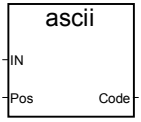
SEL	BOO	zeigt den gewählten Wert an
IN1, IN2	INT	analoge, ganzzahlige Werte
Q	INT	= value1 wenn SEL = FALSE = value2 wenn SEL = TRUE

Beschreibung:
Binärer Selektor: wählt einen Wert aus 2 ganzzahligen Werten.

(* FBS-Programm mit "SEL"-Baustein *)


```
(* ST-Äquivalenz *)
ProCmd := SEL (AutoMode, ManuCmd, InpCmd);
(* Auswahl des Prozeßbefehls *)

(* AWL-Äquivalenz *)
LD          AutoMode
SEL         ManuCmd,InpCmd
ST      ProCmd
```

ASCII



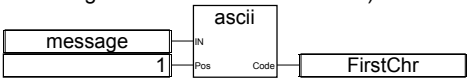
Argumente:

IN	ZK	nicht leere Zeichenkette
Pos	INT	Position des ausgewählten Zeichens im Intervall [1 .. Länge] (Länge ist die Länge der Zeichenkette IN)
Code	INT	Ascii-Code des ausgewählten Zeichens (im Intervall [0 .. 255]) 0, wenn Pos nicht in der Kette ist

Beschreibung:

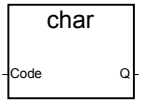
Liefert den ASCII-Code eines Zeichens in einer Zeichenkette.

(* FBS-Programm mit "ASCII"-Baustein *)




```
(* ST-Äquivalenz *)
FirstChr := ASCII (message, 1);
(* FirstChr ist der Ascii-Code des ersten Zeichens der Kette *)

(* AWL-Äquivalenz *)
LD          message
ASCII       1
ST      FirstChr
```

CHAR



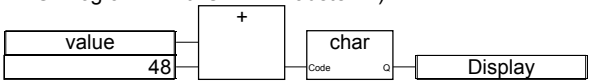
Argumente:

Code	INT	Ascii-Code im Intervall [0 .. 255]
Q	ZK	Kette mit einem Zeichen das Zeichen hat den spezifizierten Ascii-Code (der Ascii-Code wird benutzt, Modulo 256)

Beschreibung:

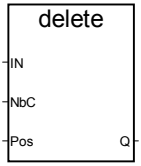
Liefert eine Zeichenkette mit einem Zeichen von einem gegebenen ASCII-Code.

(* FBS-Programm mit "CHAR"-Baustein *)




```
(* ST-Äquivalenz *)
Display := CHAR ( value + 48 );
(* value ist im Intervall [0..9] *)
(* 48 ist der Ascii-Code von '0' *)
(*das Ergebnis ist eine Kette aus 1 Zeichen von '0' bis '9' *)

(* AWL-Äquivalenz *)
LD      value
ADD     48
CHAR
ST      Display
```

DELETE

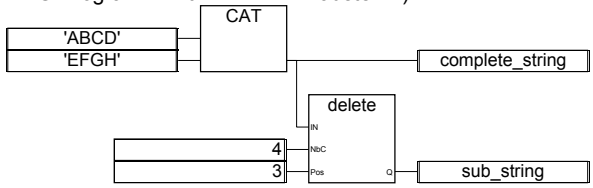


Argumente:

IN	ZK	nicht leere Zeichenkette
NbC	INT	Anzahl der zu löschenden Zeichen
Pos	INT	Position des ersten zu löschenden Zeichens (die erste gültige Pos = 1)
Q	ZK	geänderte Kette Kette leer, wenn Pos < 1 Initialkette, wenn Pos > Länge oder wenn NbC <= 0

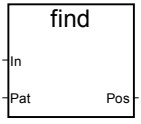
Beschreibung:
Löscht einen Teil einer Zeichenkette.

(* FBS-Programm mit "DELETE"-Baustein *)




```
(* ST-Äquivalenz *)
complete_string := 'ABCD' + 'EFGH'; (* complete_string = 'ABCDEFGH' *)
sub_string := DELETE (complete_string, 4, 3); (* sub_string = 'ABGH' *)

(* AWL-Äquivalenz *)
LD          'ABCD'
ADD         'EFGH'
ST          complete_string
DELETE      4,3
ST      sub_string
```

FIND

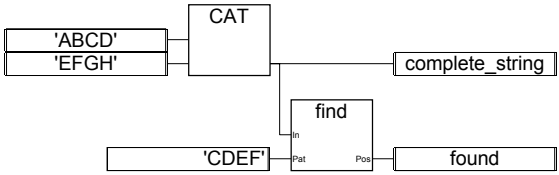


Argumente:

In	ZK	beliebige Zeichenkette
Pat	ZK	gesuchte Kette - nicht leer(Pattern)
Pos	INT	= 0 wenn Kette Pat nicht gefunden hat = Position der ersten Zeichens des ersten Vorkommens der Kette Pat (die erste gültige Position ist 1) diese Funktion unterscheidet zwischen **Groß- und Kleinbuchstaben**

Beschreibung:
Findet einen Sub-String (Kette) in einer Zeichenkette. Liefert die Position der Kette in der Zeichenkette.

(* FBS-Programm mit "FIND"-Baustein *)



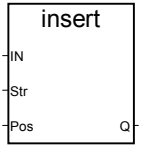
(* ST-Äquivalenz *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string = 'ABCDEFGH' *)
found := FIND (complete_string, 'CDEF'); (* found = 3 *)
```

(* AWL-Äquivalenz *)

```
LD      'ABCD'
ADD     'EFGH'
ST      complete_string
FIND    'CDEF'
ST      found
```

INSERT



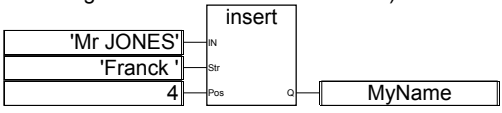
Argumente:

IN	ZK	Initialkette
Str	ZK	einzusetzende Zeichenkette
Pos	INT	Einsatzposition Einfügen erfolgt vor der Position (die erste gültige Position ist 1)
Q	ZK	geänderte Kette leere Kette, wenn Position <= 0 Verkettung beider Ketten, wenn Position größer als die ursprüngliche Länge (der Kette IN)

Beschreibung:

Einfügen eines Sub-String (Kette) an der gegebenen Position in einer Zeichenkette.

(* FBS-Programm mit "INSERT"-Baustein *)



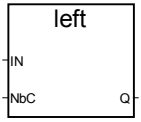
(* ST-Äquivalenz *)

```
MyName := INSERT ('Mr JONES', 'Frank ', 4);
```

(* MyName is 'Mr Frank JONES' *)

(* AWL-Äquivalenz *)
```
LD          'Mr JONES'
INSERT      'Frank ',4
ST          MyName
```

LEFT



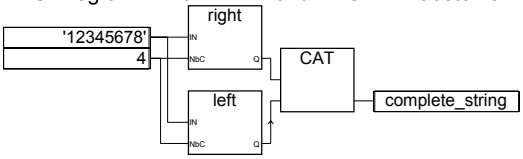
Argumente:

IN	ZK	nicht leere Zeichenkette
NbC	INT	Anzahl der herauszunehmenden Zeichen kann nicht größer als die Länge der Kette IN sein
Q	ZK	linker Teil (Länge = NbC) leere Kette NbC <= 0 Initialkette IN, wenn NbC >= IN Kettenlänge

Beschreibung:

Nimmt den linken Teil einer Zeichenkette heraus. Die Anzahl der herauszunehmenden Zeichen wird geliefert.

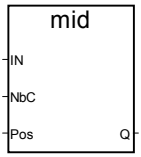
(* FBS-Programm mit "LEFT"- und "RIGHT"-Bausteinen *)



(* ST-Äquivalenz *)
```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
(* complete_string = '56781234'
der vom Aufruf RIGHT ausgegebene Wert ist '5678'
der vom Aufruf LEFT ausgegebene Wert ist '1234' *)
```

(* AWL-Äquivalenz: Zuerst erfolgt der Aufruf von LEFT *)
```
LD          '12345678'
LEFT        4
ST          sub_string  (* Zwischenergebnis *)
LD          '12345678'
RIGHT       4
ADD         sub_string
ST          complete_string
```

MID



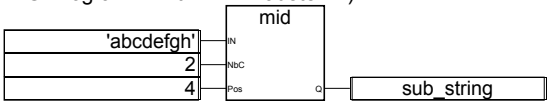
Argumente:

IN	ZK	nicht leere Zeichenkette
NbC	INT	Anzahl der herauszunehmenden Zeichen kann nicht größer als die Länge der Kette IN sein
Pos	INT	Position des ersten, herauszunehmenden Zeichens das erste Zeichen der Kette ist das Zeichen, auf das Pos zeigt (die erste gültige Position ist 1)
Q	ZK	herausgenommene Kette (Länge = NbC) leere Kette, wenn die Parameter ungültig sind

Beschreibung:

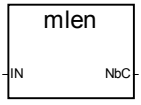
Nimmt einen Teil einer Zeichenkette heraus. Die Anzahl der herauszunehmenden Zeichen und das erste Zeichen werden geliefert.

(* FBS-Programm mit "MID"-Baustein *)




```
(* ST-Äquivalenz *)
sub_string := MID ('abcdefgh', 2, 4);
(* sub_string ist 'de' *)

(* AWL-Äquivalenz *)
LD      'abcdefgh'
MID     2,4
ST      sub_string
```

MLEN

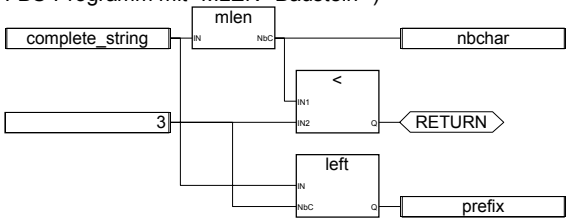


Argumente:

IN	ZK	Kette aus beliebigen Zeichen
NbC	INT	Anzahl der Zeichen in der Kette

Beschreibung:
Berechnet die Länge einer Zeichenkette.

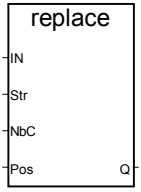
(* FBS-Programm mit "MLEN"-Baustein *)




```
(* ST-Äquivalenz *)
nbchar := MLEN (complete_string);
If (nbchar < 3) Then Return; End_if;
prefix := LEFT (complete_string, 3);
(* dieses Programm nimmt die 3 Zeichen auf der linken Seite der Kette heraus und gibt das
Ergebnis in die Zeichenketten-Variable prefix.
Nichts geschieht, wenn die Zeichenkette weniger als 3 Zeichen enthält *)
```

```
(* AWL-Äquivalenz *)
LD          complete_string
MLEN
ST          nbchar
LT          3
RETC
LD          complete_string
LEFT        3
ST          prefix
```

REPLACE



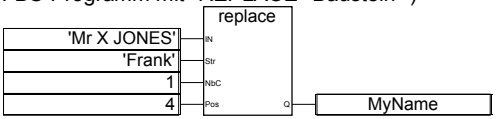
Argumente:

IN	ZK	Initialkette
Str	ZK	einzufügende Kette (um NbC Zeichen zu ersetzen)
NbC	INT	Anzahl der zu löschenden Zeichen
Pos	INT	Position des ersten geänderten Zeichens (die erste gültige Position ist 1)
Q	ZK	geänderte Kette: - NbC Zeichen gelöscht von der Position Pos - dann wird die Kette Str an dieser Position eingesetzt

ergibt eine leere Kette, wenn Pos <= 0
ergibt eine Verkettung (IN+Str), wenn Pos größer als die Länge der Kette IN ist
liefert Initialkette IN, wenn NbC <= 0

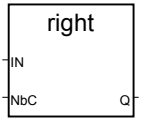
Beschreibung:
Ersetzt einen Teil einer Zeichenkette durch eine neue Zeichenfolge.

(* FBS-Programm mit "REPLACE"-Baustein *)




```
(* ST-Äquivalenz *)
MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);
(* MyName ist 'Mr Frank JONES' *)

(* AWL-Äquivalenz *)
LD          'Mr X JONES'
REPLACE     'Frank',1,4
ST          MyName
```

RIGHT

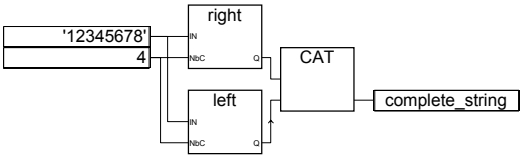


Argumente:

IN	ZK	nicht leere Zeichenkette
NbC	INT	Anzahl der herauszunehmenden Zeichen kann nicht größer als die Länge der Kette IN sein
Q	ZK	rechts herausgenommener Teil (Länge = NbC) leere Kette, wenn NbC <= 0 Komplette Kette, wenn NbC >= Kettenlänge

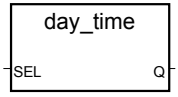
Beschreibung:
Nimmt den rechten Teil einer Zeichenkette heraus. Die Anzahl der herauszunehmenden Zeichen wird geliefert.

(* FBS-Programm mit "LEFT"- und "RIGHT"-Bausteinen *)




```
(* ST-Äquivalenz *)
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
(* complete_string ist '56781234'
der vom Aufruf RIGHT gegebene Wert ist '5678'
der vom Aufruf LEFT gegebene Wert ist '1234' *)

(* AWL-Äquivalenz: Zuerst erfolgt der Aufruf an LEFT *)
LD          '12345678'
LEFT        4
ST          sub_string  (* Zwischenergebnis *)
LD          '12345678'
RIGHT       4
ADD         sub_string
ST          complete_string
```

DAY_TIME



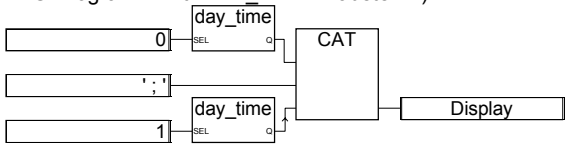
Argumente:

SEL	INT	Auswahl der Information 0= aktuelles Datum 1= aktuelle Zeit 2= Wochentag
Q	ZK	Datum/Zeit in Textform 'JJJJ/MM/TT' wenn SEL = 0 'HH:MM:SS' wenn SEL = 1 Wochentag, wenn SEL = 2 ('Monday') (Wochentage stets in Englisch)

Beschreibung:

Liefert das aktuelle Datum oder die aktuelle Uhrzeit als Zeichenkette.

(* FBS-Programm mit "DAY_TIME"-Baustein *)



```
(* ST-Äquivalenz: *)
Display := Day_Time (0) + ' ; ' + Day_Time (1);
(* Text hat stets das Format: 'JJJJ/MM/TT ; HH:MM:SS' *)

(* AWL-Äquivalenz: Zuerst erfolgt der Aufruf an day_time(1) *)
LD          1
DAY_TIME
ST          hour_str    (* Zwischenergebnis *)
LD          0
DAY_TIME
ADD         ' ; '
ADD         hour_str
ST          Display
```

ARCREATE

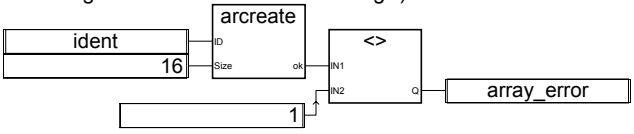
Argumente:

ID	INT	Tabellen-Identifizierer (muß im Intervall [0..15] liegen)
Size	INT	Anzahl der zu erstellenden Tabellenelemente
ok	INT	Ausführungsprotokoll : 1 = OK 2 = Identifizierer ungültig oder Tabelle bereits erstellt 3 = Größe ungültig 4 = nicht genug Speicher

Beschreibung:
Erstellung einer Ganzzahl-Tabelle.

Achtung: In einer Anwendung können höchstens **16** Tabellen erstellt werden. Sie enthalten **analoge, ganzzahlige** Werte. Da der Speicherplatz dynamisch zugeordnet wird, kann diese Funktion das System beschädigen oder fatale Fehler herbeiführen, wenn die Tabelle zuviel des verfügbaren Speicherraums in Anspruch nimmt.

(* FBS-Programm mit Tabellen-Verwaltung *)




```
(* ST-Äquivalenz *)
array_error := (ARCREATE (ident, 16) <> 1));
```

```
(* AWL-Äquivalenz *)
LD          ident
ARCREATE    16
NE          1
ST          array_error
```

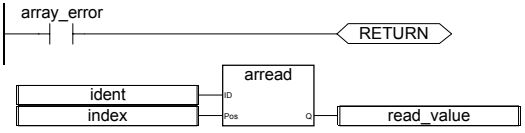
ARREAD

Argumente:

ID	INT	Tabellen-Identifizierer (muß im Intervall [0..15] liegen)
Pos	INT	Position des Elements in der Tabelle im Intervall [0 .. Größe-1]
value	INT	Wert des Elements 0, wenn die Argumente ungültig sind

Beschreibung:
Liest ein Element in einer Ganzzahl-Tabelle.

(* FBS-Programm mit Tabellen-Verwaltung *)




```
(* ST-Äquivalenz *)
If (array_error) Then Return; End_if;
read_value := ARREAD (ident, index);
(* array_error kommt vom Aufruf ARCREATE *)
```

```
(* AWL-Äquivalenz *)
LD          array_error
RETC
LD          ident
ARREAD      index
ST          read_value
```

ARWRITE

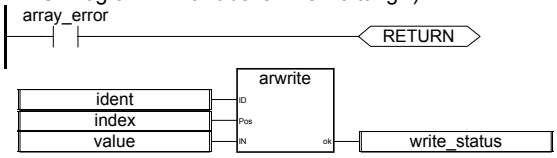
Argumente:

ID	INT	Tabellen-Identifizierer (muß im Intervall [0..15] liegen)
Pos	INT	Position des Elements in der Tabelle im Intervall [0 .. Größe-1]
IN	INT	neuer Wert für das Element
ok	INT	Ausführungsprotokoll : 1 = OK 2 = Identifizierer ungültig 3 = Pos ungültig

Beschreibung:

Speichert (schreibt) einen Wert in einer Ganzzahl-Tabelle.

(* FBS-Programm mit Tabellen-Verwaltung *)




```
(* ST-Äquivalenz *)
If (array_error) Then Return; End_if;
write_status := ARWRITE (Ident, Index, value);
(* array_error kommt vom Aufruf ARCREATE *)

(* AWL-Äquivalenz *)
LD          array_error
RETC
LD          ident
ARWRITE     index,value
ST          write_status
```

F_ROPEN

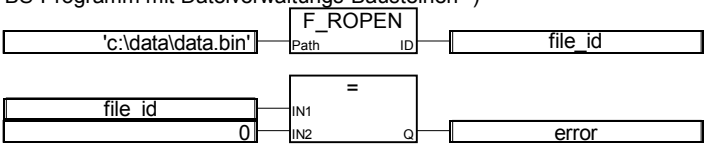
Argumente:

Path	MSG	Dateiname kann den Zugriffspfad zur Datei einschließen, mit den Zeichen \ oder / , um ein Verzeichnis zu spezifizieren. Um

		die Datenaustauschbarkeit zu erleichtern, sind / und \ äquivalent.
ID	INT	Dateinummer 0, wenn ein Fehler auftritt: Datei existiert nicht.

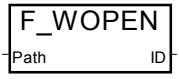
Beschreibung:
Öffnet eine Binärdatei im Lese-Modus. Zur Benutzung mit FX_READ und F_CLOSE.
Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

(* FBS-Programm mit Dateiverwaltungs-Bausteinen *)




```
(* ST-Äquivalenz: *)
file_id := F_ROPEN('c:\data \data.bin');
error := (file_id=0);

(* AWL-Äquivalenz: *)
LD          'c:\data\data.bin'
F_ROPEN
ST          file_id
EQ          0
ST          error
```

F_WOPEN

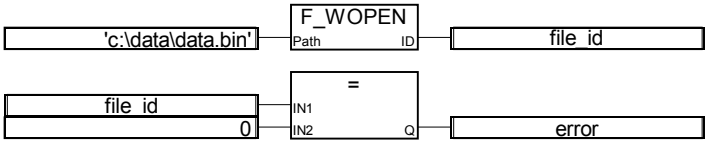


Argumente:

Path	MSG	Dateiname kann den Zugriffspfad zur Datei einschließen, mit den Zeichen \ oder / , um ein Verzeichnis zu spezifizieren. Um die Datenaustauschbarkeit zu erleichtern, sind / und \ äquivalent.
ID	INT	Dateinummer 0, wenn ein Fehler auftritt. Wenn die Datei schon existiert, wird sie überschrieben.

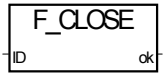
Beschreibung:
Öffnet eine Binärdatei im Schreib-Modus. Zur Benutzung mit FX_WRITE und F_CLOSE.
Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

(*FBS-Programm mit Dateiverwaltungs-Bausteinen *)




```
(* ST-Äquivalenz: *)
file_id := F_WOPEN('c:\data \data.bin');
error := (file_id=0);

(* AWL-Äquivalenz: *)
LD              'c:\data\data.bin'
F_WOPEN
ST              file_id
EQ              0
ST              error
```

F_CLOSE



Argumente:

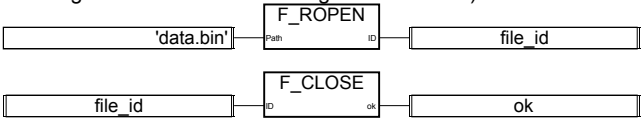
ID	INT	Dateinummer: Rückgabe von F_ROPEN oder F_WOPEN.
ok	BOO	Rückgabestatus TRUE, wenn Datei Schließen OK ist FALSE, wenn ein Fehler aufgetreten ist

Beschreibung:

Schließt eine Binärdatei, die mit den Funktionen F_ROPEN oder F_WOPEN geöffnet wurde.

Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

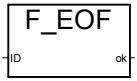
(*FBS-Programm mit Dateiverwaltungs-Bausteinen *)




```
(* ST-Äquivalenz: *)
file_id := F_ROPEN('data.bin');
ok := F_CLOSE(file_id);

(* AWL-Äquivalenz: *)
LD              'data.bin'
F_ROPEN
ST              file_id
```

```
F_CLOSE                 (* file_id ist schon im aktuellen AWL-Ergebnis *)
ST          ok
```

F_EOF



Argumente:

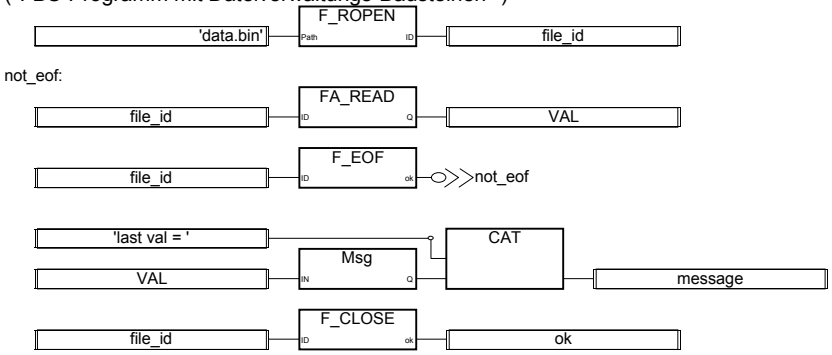
ID	INT	Dateinummer: Rückgabe von F_ROPEN oder F_WOPEN.
ok	BOO	zeigt Dateiende an TRUE, wenn das Ende der Datei beim letzten Aufruf eines Lese- oder Schreibvorgangs erreicht wurde. Mit FM_READ kann die zuetzt gelesene Zeichenkette einer Datei falsch sein, wenn das letzte Zeichen kein Zeichenketten-Endzeichen ist.

Beschreibung:

Testet, ob das Dateiende erreicht ist.

Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

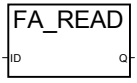
(*FBS-Programm mit Dateiverwaltungs-Bausteinen *)




```
(* ST-Äquivalenz: *)
file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
        VAL := FA_READ(file_id);
END_WHILE;
MESSAGE := 'last val = ' + msg(VAL);
ok := F_CLOSE(file_id);

(* AWL-Äquivalenz: *)
                LD          'data.bin'
                F_ROPEN
```

```
                ST          file_id
                LD          file_id
                F_EOF
                JMPC        END_OF_FILE
NOT_EOF:        LD          file_id
                FA_READ
                ST          VAL
                LD          file_id
                F_EOF
                JMPNC       NOT_EOF  (* wenn nicht eof, weiter lesen *)
END_OF_FILE:    LD          VAL
                MSG
                ST          val_msg     (* Umrechnung von VAL in eine Zeichenkette *)
                LD          'last val = '
                ADD         val_msg
                ST          MESSAGE
                LD          file_id
                F_CLOSE
                ST          ok
```

FA_READ



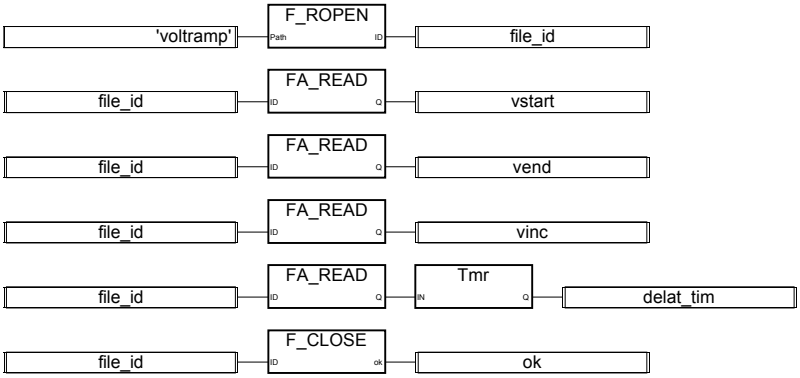
Argumente:

ID	INT	Dateinummer: Rückgabe von F_ROPEN.
Q	INT	ganzzahliger analoger Wert aus Datei gelesen

Beschreibung:

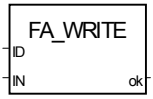
Ließt ANALOGE Variablen aus einer Binärdatei. Zur Benutzung mit F_ROPEN und F_CLOSE.
Diese Prozedur erstellt einen sequentiellen Zugriff auf die Datei von der vorherigen Position aus.
Der erste Aufruf nach F_ROPEN liest die ersten 4 Bytes der Datei,
jeder Aufruf puscht den Lese-Pointer.
Um zu prüfen, ob das Dateiende erreicht ist, benutzen Sie F_EOF.
Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

(*FBS-Programm mit Dateiverwaltungs-Bausteinen *)




```
(* ST-Äquivalenz: *)
file_id := F_ROPEN('voltramp.bin');
vstart := FA_READ(file_id);
vend := FA_READ(file_id);
vinc := FA_READ(file_id);
delta_tim := tmr(FA_READ(file_id));
ok := F_CLOSE(file_id);

(* AWL-Äquivalenz: *)
                LD          'voltramp.bin'
                F_ROPEN
                ST          file_id
                FA_READ                 (* vstart lesen *)
                ST          vstart
                LD          file_id
                FA_READ                 (*vend lesen *)
                ST          vend
                LD          file_id
                FA_READ                 (*vinc lesen *)
                ST          vinc
                LD          file_id
                FA_READ                 (*delta_tim lesen *)
                TMR                     (* Umrechnung in Timer *)
                ST          delta_tim
                LD          file_id
                F_CLOSE
                ST          ok
```

FA_WRITE



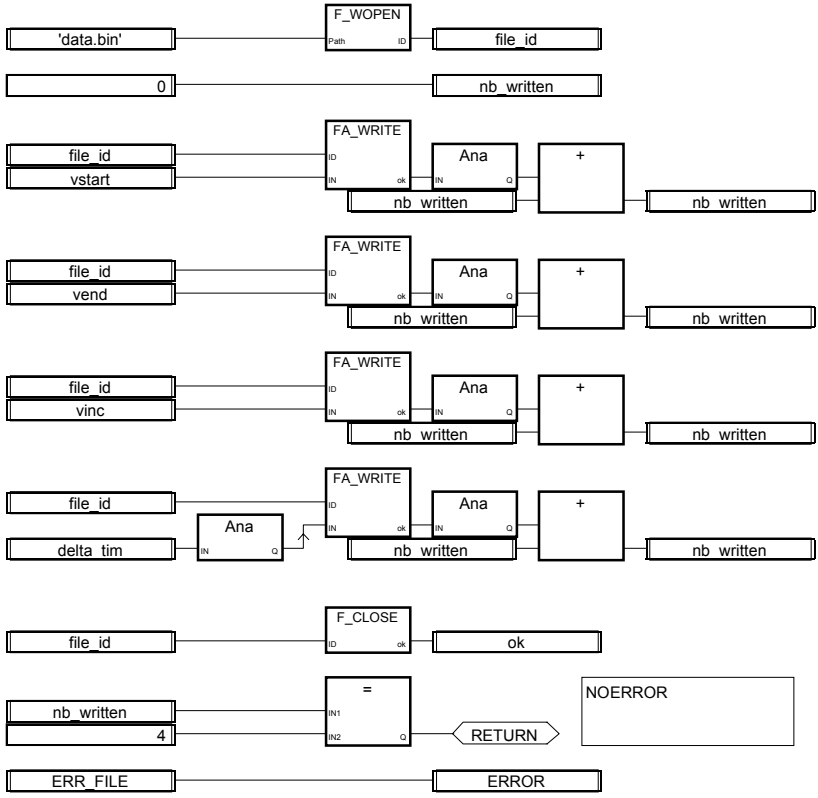
Argumente:

ID	INT	Dateinummer: Rückgabe von F_WOPEN.
IN	INT	Ganzzahliger analoger in der Datei zu schreibender Wert
OK	BOO	Ausführungsstatus: TRUE, wenn ok

Beschreibung:

Schreibt ANALOGE Variablen in eine Binärdatei.
Diese Prozedur erstellt einen sequentiellen Zugriff auf die Datei von der vorherigen Position aus.
Der erste Aufruf an F_WOPEN schreibt die ersten 4 Bytes der Datei,
jeder Aufruf puscht den Schreib-Pointer.
Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

(* FBS-Programm *)

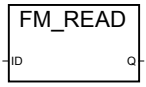


(* ST-Äquivalenz: *)

```
file_id := F_WOPEN('voltramp.bin');
nb_written  := 0;
```

```
nb_written := nb_written + ana(FA_WRITE(file_id,vstart));
nb_written := nb_written + ana(FA_WRITE(file_id,vend));
nb_written := nb_written + ana(FA_WRITE(file_id,vinc));
nb_written := nb_written + ana(FA_WRITE(file_id,ana(delta_tim)));
ok := F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
        ERROR := ERR_FILE;
END_IF;

(* AWL-Äquivalenz: *)
                LD          'voltramp.bin'
                F_ROPEN
                ST          file_id
                LD          0
                ST          nb_written
                LD          file_id     (*vstart schreiben*)
                FA_WRITE vstart
                ANA
                ADD         nb_written
                ST          nb_written
                LD          file_id     (*vend schreiben *)
                FA_WRITE vend
                ANA
                ADD         nb_written
                ST          nb_written
                LD          file_id     (*vinc schreiben *)
                FA_WRITE vinc
                ANA
                ADD         nb_written
                LD                      (*delta_tim schreiben *)
                ANA                     (* zu ganzzahligem Wert umrechnen *)
                ST          ana_delta_tim
                LD          file_id
                FA_WRITE ana_delta_tim
                ANA
                ADD         nb_written
                ST          nb_written
                F_CLOSE
                ST          ok
                LD          nb_written
                EQ          4
                RETC                    (* zurückgeben, wenn gleich 4 *)
                LD          ERR_FILE    (* sonst Fehler *)
                ST          ERROR
```

FM_READ



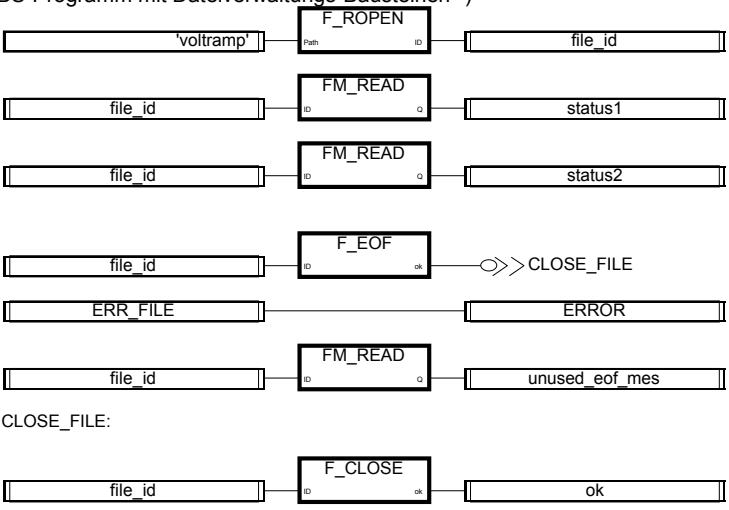
Argumente:

ID	INT	Dateinummer: Rückgabe von F_ROPEN.
Q	MSG	Zeichenketten-Wert aus Datei gelesen

Beschreibung:

Liest ZEICHENKETTEN-Variable aus einer Binärdatei.
Zu benutzen mit F_ROPEN und F_CLOSE.
Diese Prozedur erstellt einen sequentiellen Zugriff auf die Datei von der vorherigen Position aus.
Der erste Aufruf an F_ROPEN liest die erste Zeichenkette der Datei,
jeder Aufruf puscht den Lese-Pointer.
Eine Zeichenkette endet in (0), Zeilenende ('\n') oder Return ('\r');
Um zu prüfen, ob das Ende der Datei erreicht ist, benutzen Sie F_EOF.
Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

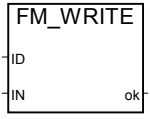
(*FBS-Programm mit Dateiverwaltungs-Bausteinen *)




```
(* ST-Äquivalenz: *)
file_id := F_ROPEN('voltramp.bin');
status1 := FM_READ(file_id);
status2 := FM_READ(file_id);
IF (F_EOF(file_id)) THEN
        ERROR := ERR_FILE;
        unused_eof_mes := FM_READ(file_id);
END_IF;
ok := F_CLOSE(file_id);

(* AWL-Äquivalenz: *)
                LD          'voltramp.bin'
                F_ROPEN
                ST          file_id
```

```
            FM_READ         (*Status1 lesen*)
            ST      status1
            LD      file_id
            FM_READ         (*Status2 lesen *)
            ST      status2
            LD      file_id
            F_EOF
            JMPNC   CLOSE_FILE (*wenn Dateiende, kein Sprung *)
            LD      ERR_FILE
            ST      ERROR
            LD      file_id
            FM_READ         (*unused_eof_mes lesen*)
            ST      unused_eof_mes
CLOSE_FILE  LD      file_id
            F_CLOSE
            ST      ok
```

FM_WRITE



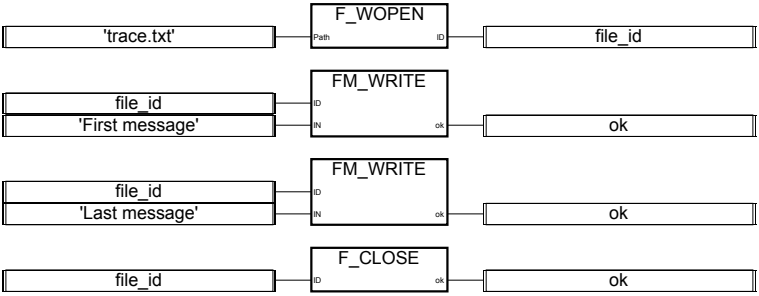
Argumente:

ID	INT	Dateinummer: Rückgabe von F_WOPEN.
IN	MSG	Zeichenketten-Wert in die Datei zu schreiben
ok	BOO	Ausführungsstatus TRUE, wenn erfolgreich

Beschreibung:

Schreibt ZEICHENKETTEN-Variable in eine Binärdatei.
Zu benutzen mit F_WOPEN und F_CLOSE.
Eine Zeichenkette wird in die Datei als eine in null endende Zeichenkette geschrieben.
Diese Prozedur erstellt einen sequentiellen Zugriff auf die Datei von der vorherigen Position aus.
Der erste Aufruf an F_WOPEN schreibt die erste Zeichenkette der Datei,
jeder Aufruf puscht den Schreib-Pointer.
Diese Funktion ist nicht im ISaGRAF Simulator vorhanden.

(*FBS-Programm mit Dateiverwaltungs-Bausteinen *)

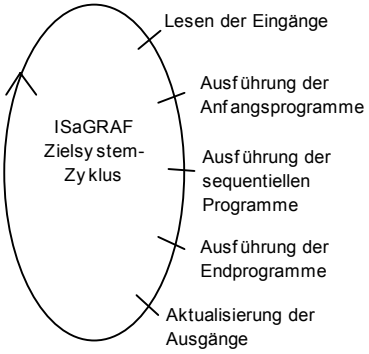



```
(* ST-Äquivalenz: *)
file_id := F_WOPEN('trace.txt');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);

(* AWL-Äquivalenz: *)
                LD          'trace.txt'
                F_WOPEN
                ST          file_id
                FM_WRITE    'First message'     (*first msg schreiben *)
                ST          ok
                LD          file_id
                FM_WRITE    'Last message'      (*second msg schreiben *)
                ST          ok
                LD          file_id
                F_CLOSE
                ST          ok
```

C. Zielsystem Benutzerhandbuch

C.1. Einleitung

Das ISaGRAF Zielsystem ist eine Echtzeit-Software, die eine ISaGRAF Anwendung auf Ihrem Industrierechner oder Ihrer Karte entsprechend des folgenden, bekannten Schemas betreibt:



Ein Zielsystem-Zyklus besteht aus dem Ablesen der physischen Eingänge des auszuführenden Prozesses, der Verarbeitung der Anwendungsdaten gemäß der Anwendungsprogramme der ISaGRAF Workstation[1] und der Aktualisierung der physischen Ausgänge.

- Der erste Teil dieses Abschnitts erläutert die Inbetriebnahme spezifischer Zielsysteme (DOS, OS-9, VxWorks und NT). Für jedes dieser Systeme wird zuerst beschrieben, wie das ISaGRAF Zielsystem betrieben wird. Weiter werden bestimmte Funktionalitäten erläutert, wie beispielsweise: Anlaufen des Zielsystems beim Einschalten, Fehlerverwaltung, allgemeines Verhalten, ...

- Der zweite Teil beschreibt die Implementierung von benutzerdefinierten C-Funktionen, -Funktionsbausteinen und -Umrechnungsfunktionen, mit denen die Leistung des ISaGRAF Zielsystems gesteigert werden kann.

- Der dritte Teil enthält Informationen über Modbus und die ISaGRAF Implementierung und beschreibt das Blockformat der verschiedenen Funktionscodes.

- Der vierte Teil beschreibt einige Werkzeuge für die Verwaltung von Stromausfällen und für das Wiederanlaufen des Zielsystems.

[1] Es wird vorrausgesetzt, daß die ISaGRAF Workstation dem Benutzer geläufig ist.

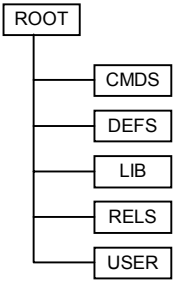
C.2. Installation

Die Installation erfordert ca. 1 Mbyte freien Platz auf Ihrer Festplatte.

Das auf der Diskette mitgelieferte install.bat installiert alle für eine spezifische Plattform benötigten Dateien auf Ihrem PC.

Beispiel: a:\install a: c:*path*
installiert Dateien vom Diskettenlaufwerk a: auf c: im Verzeichnis *path*.

Die folgende Verzeichnisarchitektur wird benutzt:



Das ROOT-Verzeichnis enthält einige Werkzeuge und Readme-Dateien
Das CMDS-Verzeichnis enthält die ausführbaren Dateien
Das DEFS-Verzeichnis enthält die Definitionsdateien
Das LIB-Verzeichnis enthält die Bibliotheken
Das RELS-Verzeichnis enthält die verschieblichen (Objekt-)Dateien
Das USER-Verzeichnis enthält benutzerdefinierte "C"-Prozeduren für C-Funktionen, -Funktionsbausteine und -Umrechnungsfunktionen (Quell- und Leitdateien)

Daraufhin kann die installierte Plattform in Betrieb genommen werden.

C.3. Inbetriebnahme des ISaGRAF DOS Zielsystems

C.3.1. Ausführung von ISaGRAF: ISA.EXE

In der MS-DOS-Implementierung wird das Zielsystem als ein einziges Programm ausgeführt: ISA.EXE
Für die Inbetriebnahme leiten Sie einfach den Hilfsbefehl isa -? im CMDS-Verzeichnis ein.

In einem solchen System können die Operationen kritisch sein. Beispielsweise sollte der Kommunikationsanschluß nicht überlastet werden, damit eine gute Leistung garantiert werden kann.
Das Zielsystemprogramm verhindert nicht die Ausführung von interruptgesteuerten Routinen.

Kommunikationsanschluß und Konfiguration: -t Option

Das ISaGRAF Zielsystem benutzt einen seriellen Anschluß für die Kommunikation mit dem Debugger. Der Name des Anschlusses wird mit Hilfe der -t Option definiert. Da die Kommunikationsschnittstelle so gestaltet wurde, daß sie mit beliebigen Geräten kompatibel ist, können die Anschlüsse COM1, COM2 oder COM3 benutzt werden (abhängig von der BIOS-Version).

Kein Vorgabewert: Wenn diese Option nicht benutzt wird, ist keine Kommunikation mit dem Zielsystem möglich. In diesem Fall wird möglicherweise Fehler N° 7 angezeigt.

Die Kommunikation über einen Ethernet-Anschluß ist mit dem ISaGRAF DOS Zielsystem nicht möglich. Fragen Sie Ihren Lieferanten nach einer speziellen Implementierung.

Die Kommunikationsparameter müssen vor der Inbetriebnahme von ISaGRAF eingestellt werden, so daß der Benutzer die erforderlichen Parameter völlig frei benutzen kann. Wenn Sie den Workstation-Debugger benutzen, sollten Sie sich vergewissern, daß die Workstation-Kommunikationsparameter (siehe Benutzerhandbuch: Programmverwaltung) mit denen des Zielsystems übereinstimmen.

Beispiel:
MODE COM1:9600,N,8,1
Setzt Kommunikationsparameter auf die folgenden Werten:
- Baudgeschwindigkeit 9600
- keine Parität
- 8 Datenbits
- 1 Stoppbit

Bitte beachten Sie, daß die Workstation-Vorgabeeinstellung 19200 Baud bei manchen BIOS-Versionen nicht gestattet ist.
CJ liefert die ISAMOD.EXE Utility für die Einstellung der Workstation-Parameter:
ISAMOD COM1
Sie ist äquivalent zu MODE COM1:19200,N,8,1

Slave-Nummer: -s Option

Diese Option definiert die Slave-Nummer des Zielsystems. Sie kann zwischen 1 und 255 liegen, mit Ausnahme der Nummer 13 ($0D). Die Slave-Nummer wird vom Kommunikationsprotokoll benutzt, hauptsächlich um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme zusammengeschlossen sind. Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß der Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) mit dem des Zielsystems übereinstimmt.

Vorgabewert: Die vorgegebene Slave-Nummer ist 1 (wie die der Workstation)

Beispiele:

isamod COM1 Konfiguriert COM1 mit 19200 Baud, keine Parität, 8 Datenbits, 1 Stoppbit.

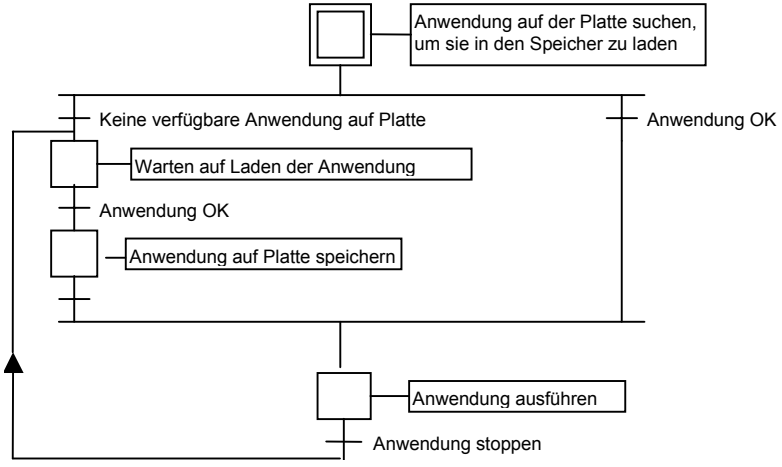
isa -t=COM1 Startet das ISaGRAF Zielsystem mit der Vorgabe-Slave-Nummer (1) und mit COM1 als Kommunikationsanschluß.

isa -s=3 -t=COM1 Startet das ISaGRAF Zielsystem mit der Slave-Nummer 3 und mit COM1 als Kommunikationsanschluß.

C.3.2. Spezifische Funktionalitäten

Starten des ISaGRAF Zielsystems

Beim Start des Zielsystems wird der folgende Algorithmus ausgeführt.



- **Definitionen**

Der Anwendungscode ist die binäre Datenbank, welche von der Workstation generiert und geladen und dann vom Zielsystem ausgeführt wird. Sie kann durch eine Symboltabelle ergänzt werden.

Die Symboltabelle einer Anwendung ist eine ASCII-Datenbank, die von der Workstation generiert und geladen wird. Diese Tabelle stellt die Verbindung zwischen Symbolobjekten und internen Zielsystemobjekten her. Sie ist im Zielsystem nicht erforderlich, außer für die Verwaltung von benutzerspezifischen Symbolen. Für weitere Informationen über Symboltabellen siehe Benutzerhandbuch: Fortgeschrittene Programmierungstechniken.

- **Anwendungssicherung**

Wenn eine neue Anwendung vom Workstation-Debugger in das Zielsystem geladen wird, wird der Anwendungscode im aktuellen Verzeichnis des Zielsystems unter dem folgenden Namen gesichert:

ISAx1 ISaGRAF Anwendungscode-Sicherungsdatei (x ist die Slave-Nummer)

Wenn die Anwendungssymboltabelle vorher geladen wurde, wird sie auch im aktuellen Verzeichnis des Zielsystems gespeichert, und zwar unter dem Namen:

ISAx6 ISaGRAF Anwendungssymbol-Sicherungsdatei (x ist die Slave-Nummer)

Beim Start des ISaGRAF Zielsystems werden diese Anwendungscode- und Anwendungssymboldateien im aktuellen Verzeichnis gesucht und in den Speicher geladen.

Wenn keine Symboldatei verfügbar ist, startet das Zielsystem die Ausführung des Anwendungscodes ohne geladene Symbole.

Wenn kein Anwendungscode verfügbar ist, wartet das Zielsystem auf das Laden einer Anwendung.

Um das Zielsystem beim Einschalten mit einer bestimmten Anwendung zu starten, ohne den Debugger-Anschluß zu benutzen, kopiert man diese Dateien von der Platte (wenn sich die Workstation auf demselben PC befindet) oder unter Anwendung einer Diskette direkt in das aktuelle Verzeichnis des Zielsystems. Wenn Ihr Zielsystem keine Platte besitzt, können Sie eine virtuelle Platte benutzen.

Wenn die ISaGRAF Workstation im Standardverzeichnis \ISAWIN installiert ist,
ist die Anwendungscodedatei des Projekts MYPROJ:

\ISAWIN\APL\MYPROJ\appli.x8m

und die Anwendungssymboldatei des Projekts MYPROJ:

\ISAWIN\APL\MYPROJ\appli.tst

Beispiel:
Vom Verzeichnis, in dem isa.exe installiert ist, wenn der folgende Befehl eingegeben wird:

copy \ISAWIN\APL\MYPROJ\appli.x8m isa11

Dann sucht isa.exe die Anwendung ‘myproj’ und führt sie aus.

Diese Befehle können zum Beispiel in einer Batch-Datei gruppiert und dann vom Werkzeugmenü der Workstation eingeleitet werden (siehe Benutzerhandbuch: Programmverwaltung).

Fehlerverwaltung und Ausgangsmeldungen

Die ISaGRAF Zielsystem-Software ermöglicht die Verwaltung der Fehlererkennung. Im Anhang finden Sie eine Liste der Fehlermeldungen und deren Beschreibung.

Die Fehlererkennung wird folgendermaßen verarbeitet:
- Ein Fehler setzt sich aus einem Fehler und einer Argumentnummer zusammen, die zur ISaGRAF Fehlerroutine gesendet werden.
- Wenn die Fehlererkennung in den Ausführungsparametern der Workstation aktiviert wurde, wird der Fehler verarbeitet. Ansonsten gehen die Informationen verloren und die Fehlerverwaltung endet.

Bei der Verarbeitung eines Fehlers:
- Fehlernummer (Dezimalwert) und Argument (Hexadezimalwert) werden im Vorgabeausgang stdout angezeigt.
- Fehlernummer und Argument werden in einem FIFO Fehlerpuffer gespeichert, um zu einem späteren Zeitpunkt abgerufen werden zu können. Die Größe des Fehlerpuffers wird in den Ausführungsparametern der Workstation definiert. Wenn der Puffer voll ist, geht bei jedem neu eingehenden Fehler der jeweils älteste Fehler verloren.
- Fehler können entweder vom Debugger oder von der laufenden Anwendung unter Anwendung des SYSTEM-Aufrufs abgerufen werden (siehe Benutzerhandbuch).

Wenn der Debugger einen Fehler erkannt hat, erscheint eine Fehlermeldung mit der Beschreibung des Fehlers im Fehlerfenster. Je nach Anwendungskontext (in Betrieb oder nicht) zeigt der Debugger den Namen des Objekts (Variable oder Programm), in dem der Fehler aufritt, oder den Argumentfehler (Dezimalwert) in Klammern [x] an, was bei jedem Fehler eine unterschiedliche Bedeutung hat.

Eine Willkommensmeldung und die Fehlerwerte werden beim Start des Zielsystems, wenn ein Fehler erkannt wurde, im Vorgabeausgang stdout angezeigt. Wenn diese Anzeige auf dem Standard-Ausgangskanal unerwünscht ist, kann ein Umleitungsbefehl benutzt werden, wie zum Beispiel:

isa -t=COM1 -s=1 **>NUL**

Systemuhr

Da das ISaGRAF Zielsystem so entworfen wurde, daß es unter beliebigen Systemen laufen kann, wird der Standard-Tick als Zeitreferenz für die Zyklussynchronisierung und die Aktualisierung der Timer-Variablen benutzt. Er beträgt ca. 55 Millisekunden.

Daher sind die Timer-Variablen auf eine Genauigkeit von 55 ms begrenzt. Eine spezifische Zyklusdauer von weniger oder gleich 55 ms und ungleich null generiert einen Zykluszeitüberlauffehler (Fehler 62) und Zyklen, die nicht getriggert werden.

Es ist vorteilhaft, den System-Tick nicht zu modifizieren, damit residente Anwendungen oder in die Anwendung integrierte C-Funktionen und -Funktionsbausteine nicht von der ISaGRAF Ausführung beeinflußt werden.

Fragen Sie Ihren Lieferanten nach einer spezifischen Implementierung, wenn Ihre Anwendung eine größere Genauigkeit erfordert.

ISaGRAF beenden

Wenn der Benutzer eine Anwendung in nicht-industriellen Umständen auf einem PC testet, kann er ISaGRAF durch Anschlagen einer komplexen Tastenkombination stoppen. Der Gebrauch mehrerer Tasten sorgt für eine größere Sichereit und verhindert, daß ISaGRAF unerwartet beendet wird. Die folgende Tastenkombination wird benutzt:

Umschalt + Strg + Alt

Wenn die industrielle Anwendung nicht durch Anschlagen einer Taste gestoppt werden soll, sollte diese Möglichkeit deaktiviert werden.

Eine gefährliche Nebenwirkung des schnellen Beendens ist, daß die E/A-Kartenschnittstelle nicht geschlossen wird. Der richtige Weg, Ihr ISaGRAF Zielsystem zu beenden, ist folgender:
- die Anwendung vom Debugger aus stoppen (die E/A-Karten werden geschlossen)
- das ISaGRAF Zielsystem mit Hilfe der Tastatur stoppen.

Anwendungsgröße

Da das ISaGRAF MS-DOS Zielsystem für Intel Real Mode entworfen wurde, beträgt die Maximalgröße einer Datenstruktur 64K. Daher sollte der von der Workstation geladene Anwendungscode diese Grenze nicht überschreiten. In äußerst seltenen Fällen kann es vorkommen, daß auch die von ISaGRAF zugewiesene interne Struktur diese Grenze überschreitet und Ihre Anwendung nach dem Laden zusammenbricht. Der insgesamt verfügbare Speicherplatz ist auf 640K konventionellen Speicher begrenzt.

Fragen Sie Ihren Lieferanten nach einer speziellen Implementierung, falls Ihre Anwendung eine größere Speicherkapazität benötigt.

C.4. Inbetriebnahme des ISaGRAF OS9 Zielsystems

Zuerst müssen bestimmte Dateien (zumindest die ausführbaren Dateien im CMDS-Verzeichnis) mit Hilfe eines beliebigen Dateiübertragungswerkzeugs auf Ihr OS-9-Zielsystem übertragen werden.
Für die Inbetriebnahme leiten Sie dann einfach die Hilfsbefehle aus dem CMDS-Verzeichnis Ihres OS-9 Systems ein:

```
isa -?
isaker -?
isatst -?
Isanet -?
```

C.4.1. Ausführung des ISaGRAF Einzeltasks: isa

Das ISaGRAF Zielsystem kann als Einzeltask ausgeführt werden. Die Operationen in einer solchen Konfiguration können allerdings kritisch sein. Beispielsweise sollte der Kommunikationsanschluß nicht überlastet werden, damit eine gute Leistung garantiert werden kann. Im OS-9 Multitasking-System können verschiedene ISaGRAF Einzeltask-Zielsysteme auf einer einzigen Zentraleinheit laufen, sofern sie unterschiedliche Slave-Nummern und Kommunikationsanschlüsse besitzen.
Diese Einzeltask-Implementierung wurde hauptsächlich für bestimmte Hardware-Plattformen entworfen, beispielsweise für preisgünstige Karten oder MS-DOS PCs, oder um einen ersten Prototyp zu erstellen, wenn er auf eine neue Plattform portiert wird. Deshalb sollte die Multitasking-Implementierung des ISaGRAF Zielsystems bevorzugt werden.
Das ISaGRAF Einzeltask-Zielsystem verhindert nicht die Ausführung von Hintergrundverarbeitungen oder interruptgesteuerten Routinen.

Kommunikationsverbindung und Konfiguration: -t Option

Das ISaGRAF Einzeltask-Zielsystem benutzt einen seriellen Anschluß für die Kommunikation mit dem Debugger. Der Name des Beschreibers wird mit der -t Option spezifiziert.
Kein Vorgabewert: Wenn diese Option nicht benutzt wird, ist keine Kommunikation mit dem Zielsystem möglich. In diesem Fall wird möglicherweise Fehler N° 7 angezeigt.

Die Kommunikation über einen Ethernet-Anschluß ist mit der Einzeltasking-Implementierung nicht möglich.

Eine serielle Verbindungseinheit wird im binären Datenübertragungsmodus geöffnet (keine Steuerzeichen, kein XON/XOFF). Die anderen Kommunikationsparameter werden vor der Inbetriebnahme von ISaGRAF eingestellt, so daß der Benutzer die erforderlichen Parameter völlig frei benutzen kann. Wenn Sie den Workstation-Debugger benutzen, sollten Sie sich vergewissern, daß die Workstation-Kommunikationsparameter (siehe Benutzerhandbuch: Programmverwaltung) mit denen des Zielsystems übereinstimmen.
Beispiel:
xmode /t0 baud=19200
Setzt die Kommunikationsgeschwindigkeit auf 19200 Baud auf /t0 Einheit

Slave-Nummer: -s Option

Diese Option definiert die Slave-Nummer des Zielsystems. Sie kann zwischen 1 und 255 liegen, mit Ausnahme der Nummer 13 ($0D). Die Slave-Nummer wird im Kommunikationsprotokoll benutzt, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme laufen. Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß der Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) mit dem des Zielsystems übereinstimmt.

Vorgabewert: Die vorgegebene Slave-Nummer ist 1 (wie die der Workstation)

Beispiele:

isa -t=/t0 Startet ein ISaGRAF Einzeltask-Zielsystem mit der vorgegebenen Slave-Nummer (1) und mit /t0 als Kommunikationsanschluß.

isa -s=3 -t=/t1 Startet ein ISaGRAF Einzeltask-Zielsystem mit Slave-Nummer 3 und mit /t1 als Kommunikationsanschluß.

isa -t=/t0 &
isa -s=3 -t=/t1 Startet zwei ISaGRAF Einzeltask-Zielsysteme. Eins mit der vorgegebenen Slave-Nummer (1) und mit /t0 als Kommunikationsanschluß. Das andere mit der Slave-Nummer 3 und mit /t1 als Kommunikationsanschluß.

C.4.2. Ausführung der ISaGRAF Multitasks: isaker, isatst, isanet

Um eine schnellere Antwortzeit des ISaGRAF Zielsystemkernels und der Kommunikationsverbindung zu erzielen, wird das Zielsystem in zwei Tasks aufgespalten, die die Kommunikationsarbeit (Kommunikations-Task isatst oder isanet) von der Anwendungsausführung (isaker kernel task) trennen.
Eine solche Architektur ist weitaus flexibler, da sie dem Benutzer ermöglicht, mehrere, mit einem einzigen Kernel-Task verbundene Kommunikations-Tasks oder bis zu 4 Kernels mit einem einzigen Kommunikations-Task auszuführen. Dies erleichtert bestimmte Integrationen, beispielsweise eine Prozeßvisualisierungsverbindung und die Workstation-Debugger-Verbindung auf einer einzigen Anwendung oder eine Einzelverbindung mit 4 verschiedenen Anwendungen über einen einzigen physischen Anschluß.

Die Kernel- und Kommunikations-Tasks sind unabhängig voneinander und können getrennt gestartet werden. Es ist lediglich erforderlich, den Kernel-Task (oder Tasks) zuerst zu starten, damit er seine Systemumgebung initialisieren kann und die Kommunikations-Tasks verbunden werden können.

Das ISaGRAF Multitask-System verhindert nicht die Ausführung von Hintergrundverarbeitungen oder interruptgesteuerten Routinen.

C.4.2.1. Ausführung des Kernel-Tasks: isaker

Slave-Nummer: -s Option

Diese Option definiert die Kernel-Slave-Nummer des Zielsystems. Sie kann zwischen 1 und 255 liegen, mit Ausnahme der Nummer 13 ($0D). Die Slave-Nummer wird im Kommunikationsprotokoll und von dem (den) mit dem Kernel verbundenen Task(s) benutzt. Sie wird gebraucht, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme laufen.

Vorgabewert: Die vorgegebene Slave-Nummer ist 1 (wie die der Workstation)

C.4.2.2. Ausführung des seriellen Kommunikations-Tasks: isatst

Kommunikationsverbindung und Konfiguration: -t Option

Der Zielsystem-Kommunikations-Task isatst benutzt eine serielle Verbindung für die Kommunikation mit dem Debugger. Der Name des Beschreibers wird mit der -t Option spezifiziert.

Kein Vorgabewert: Wenn diese Option nicht benutzt wird, ist keine Kommunikation mit dem Zielsystem möglich. In diesem Fall wird möglicherweise Fehler N° 7 angezeigt.

Die Kommunikation über einen Ethernet-Anschluß ist mit der isatst Task-Implementierung nicht möglich.

Die serielle Verbindungseinheit wird im binären Datenübertragungsmodus geöffnet (keine Steuerzeichen, kein XON/XOFF). Die anderen Kommunikationsparameter müssen vor der Inbetriebnahme von ISaGRAF eingestellt werden, so daß der Benutzer die erforderlichen Parameter völlig frei benutzen kann. Wenn Sie den Workstation-Debugger benutzen, sollten Sie sich vergewissern, daß die Workstation-Kommunikationsparameter (siehe Benutzerhandbuch: Programmverwaltung) mit denen des Zielsystems übereinstimmen.
Beispiel:
xmode /t0 baud=19200
Setzt die Kommunikationsgeschwindigkeit auf 19200 Baud auf /t0 Einheit

Slave-Nummer: -s Option

Diese Option spezifiziert die Zielsystem-Kernel-Slave-Nummer(n), mit der (denen) der Kommunikations-Task verbunden ist. Sie liegt zwischen 1 und 255, mit Ausnahme der Nummer 13 ($0D). Diese Option kann bis zu viermal wiederholt werden, um eine Verbindung mit bis zu 4 verschiedenen Kernel-Slaves herzustellen. Die Slave-Nummer wird im Kommunikationsprotokoll benutzt, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme laufen. Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß der Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) mit einem bestehenden Zielsystem übereinstimmt (Kernel- und Kommunikations-Tasks).

Vorgabewert: Die vorgegebene Slave-Nummer ist 1 (wie die der Workstation)

Logische Nummer des Kommunikations-Tasks: -c Option

Diese Option spezifiziert die logische Nummer des Kommunikations-Tasks, die für die gleichzeitige Verwaltung mehrerer Kommunikations-Tasks benutzt wird. Sie liegt zwischen 1 und 255 und muß für jeden Kommunikations-Task unterschiedlich sein.

Vorgabewert: Die zuletzt spezifizierte -s Option wird benutzt. Der Vorgabewert sichert die Kompatibilität mit älteren ISaGRAF Versionen (3.0).

C.4.2.3. Ausführung des Ethernet-Kommunikations-Tasks: isanet

Kommunikationsverbindung und Konfiguration: -t Option

Der Zielsystem-Kommunikations-Task isanet benutzt die Standard-Ethernet-Verbindung für die Kommunikation mit dem Debugger. Die Anschlußnummer wird mit der -t Option spezifiziert.

Kein Vorgabewert: Wenn diese Option nicht benutzt wird, ist keine Kommunikation mit dem Zielsystem möglich. In diesem Fall wird möglicherweise Fehler N° 7 angezeigt.

Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß die Kommunikations-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) mit denen des Zielsystems übereinstimmen.

Für ISaGRAF ist das OS-9-Zielsystem der Server. Der Debugger ist der Client, der sich an die spezifizierte Anschlußnummer anschließt.
Bevor Sie Ihre erste Debugging-Sitzung auf Ethernet starten, vergewissern Sie sich, daß Ihre OS-9 Ethernet-Einrichtung richtig konfiguriert ist. Sie können beispielsweise ein "Ping" an das OS-9-System senden.

Slave-Nummer: -s Option

Diese Option spezifiziert die Zielsystem-Kernel-Slave-Nummer(n), mit der (denen) der Kommunikations-Task verbunden ist. Sie liegt zwischen 1 und 255, mit Ausnahme der Nummer 13 ($0D). Diese Option kann bis zu viermal wiederholt werden, um eine Verbindung mit bis zu 4 verschiedenen Kernel-Slaves herzustellen. Die Slave-Nummer wird im Kommunikationsprotokoll benutzt, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme laufen. Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß der Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) mit einem bestehenden Zielsystem übereinstimmt (Kernel- und Kommunikations-Tasks).

Vorgabewert: Die vorgegebene Slave-Nummer ist 1 (wie die der Workstation)

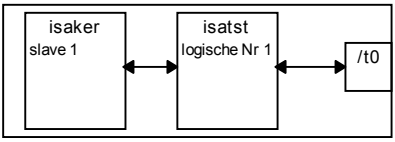
Logische Nummer des Kommunikations-Tasks: -c Option

Diese Option spezifiziert die logische Nummer des Kommunikations-Tasks, die für die gleichzeitige Verwaltung mehrerer Kommunikations-Tasks benutzt wird. Sie liegt zwischen 1 und 255 und muß für jeden Kommunikations-Task unterschiedlich sein.

Vorgabewert: Die zuletzt spezifizierte -s Option wird benutzt. Der Vorgabewert sichert die Kompatibilität mit älteren ISaGRAF Versionen (3.0).

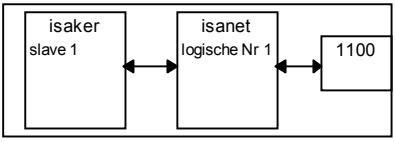
C.4.2.4. Beispiele:

```
isaker &
isatst -t=/t0
```



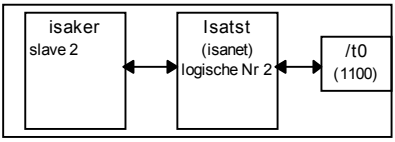
Startet:
Einen ISaGRAF Kernel-Task mit Vorgabe-Slave Nummer (1).
Einen seriellen ISaGRAF Kommunikations-Task auf /t0 com Anschluß, verbunden mit Vorgabe-Slave Nummer (1) und mit vorgegebener logischer Nummer (zuletzt spezifizierte Slave-Nummer = Vorgabe = 1).

isaker &
isanet -t=1100



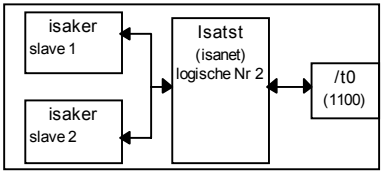
Startet:
Einen ISaGRAF Kernel-Task mit Vorgabe-Slave Nummer (1).
Einen ISaGRAF Ethernet-Kommunikations-Task auf Anschluß Nummer 1100, verbunden mit Vorgabe-Slave Nummer (1) und mit vorgegebener logischer Nummer (zuletzt spezifizierte Slave-Nummer = Vorgabe = 1).

isaker -s=2 &
isatst -t=/t0 -s=2 (bzw. isanet -t=1100 -s=2)



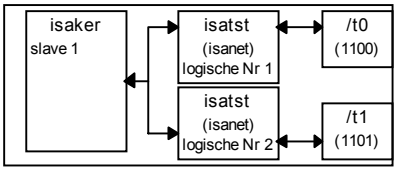
Startet:
Einen ISaGRAF Kernel-Task mit Slave Nummer 2.
Einen seriellen (Ethernet) ISaGRAF Kommunikations-Task auf /t0 com Anschluß (Anschlußnummer 1100) verbunden mit Slave Nummer 2 und mit vorgegebener logischer Nummer (zuletzt spezifizierte Slave-Nummer = 2).

Isaker -s=1 &
isaker -s=2 &
isatst -t=/t0 -s=1 -s=2 (bzw. isanet -t=1100 -s=1 -s=2)



Startet:
Einen ISaGRAF Kernel-Task mit Slave Nummer 1.
Einen ISaGRAF Kernel-Task mit Slave Nummer 2.
Einen seriellen (Ethernet) ISaGRAF Kommunikations-Task auf /t0 com Anschluß (Anschlußnummer 1100) verbunden mit Slaves Nummer 1 und 2 und mit vorgegebener logischer Nummer (zuletzt spezifizierte Slave-Nummer = 2).

Isaker -s=1 &
isatst -t=/t0 -s=1 -c=1 & (bzw. isanet -t=1100 -s=1 -c=1 &)
isatst -t=/t1 -s=1 -c=2 (bzw. isanet -t=1101 -s=1 -c=2)



Startet:
Einen ISaGRAF Kernel-Task mit Slave Nummer 1.
Einen seriellen (Ethernet) ISaGRAF Kommunikations-Task auf /t0 com Anschluß (Anschlußnummer 1100) verbunden mit Slave Nummer 1 und mit logischer Nummer 1.
Einen seriellen (Ethernet) ISaGRAF Kommunikations-Task auf /t0 com Anschluß (Anschlußnummer 1100) verbunden mit Slave Nummer 1 und mit logischer Nummer 2.
Hinweis:
Serielle und Ethernet-Kommunikations-Tasks können miteinander kombiniert werden.

C.4.3. Spezielle Funktionalitäten

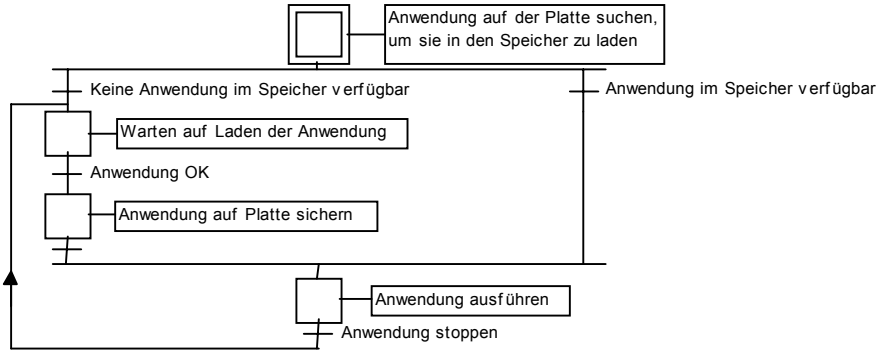
Kommunikationsverbindung

Der OS-9 Serial Character Manager ist äußerst flexibel. Fast alle bidirektionalen, von Microware unterstützten physischen Einrichtungen können benutzt werden:

Beispiel:
Die serielle Verbindung kann ein Netzwerkpfad zu einem physischen Anschluß sein, der sich auf einer anderen Zentraleinheit befindet.
In diesem Fall kann die -t Option so benutzt werden: -t=/nr/MASTER/t0
Hier wird die Kommunikationsverbindung auf eine Zentraleinheit namens MASTER in einem ramnet Netzwerk deportiert. Der benutzte physische Anschluß ist /t0.

Starten des ISaGRAF Zielsystems

Beim Start des Zielsystems wird der folgende Algorithmus ausgeführt.



- **Definitionen**

Der Anwendungscode ist die binäre Datenbank, welche von der Workstation generiert und geladen und dann vom Zielsystem ausgeführt wird. Sie kann durch die Symboltabelle ergänzt werden.
Die Symboltabelle einer Anwendung ist eine ASCII-Datenbank, die von der Workstation generiert und geladen wird. Diese Tabelle stellt die Verbindung zwischen Symbolobjekten und internen Zielsystemobjekten her. Sie ist im Zielsystem nicht erforderlich, außer für die Verwaltung von benutzerspezifischen Symbolen. Für weitere Informationen über Symboltabellen siehe Benutzerhandbuch: Fortgeschrittene Programmierungstechniken.

- **ISaGRAF OS-9 Objekte und Multianwendung**

Alle Namen eines allgemeinen ISaGRAF Objekts beginnen mit '**ISAxn**', wobei **x** die Kernel-Slave-Nummer und **n** eine Platznummer mit einer bestimmten Bedeutung ist, mit Ausnahme von **ISAy3** wobei **y** die logische Nummer des Kommunikations-Tasks in der Multitasking-Implementierung ist.
Verschiedene Anwendungen (Kernel- und Kommunikationstasks) können auf einer Zentraleinheit gleichzeitig laufen, sofern sie jeweils unterschiedliche Slave-Nummern und logische Kommunikationstask-Nummern haben. Bei der Ausführung verschiedener Anwendungen sollte der Benutzer jedoch bei geteiltem Zugriff bestimmter Anwendungsobjekte, wie E/A-Karten, äußerst vorsichtig handeln. Verschiedene Anwendungen (Kernels) können beispielsweise getrennte physische Karten benutzen, außer es wird eine Art E/A-Server oder Semaphor durch den E/A-Treiber implementiert.

OS-9 Objektnamen:
Plattendateien:

ISAx1	ISaGRAF Anwendungscode-Sicherungsdatei
ISAx6	ISaGRAF Anwendungssymbol-Sicherungsdatei

Speichermodule:

ISAx0	ISaGRAF Kernel-Systemdaten
ISAx1	ISaGRAF Anwendungscode

ISAx2	ISaGRAF Kernel-Echtzeitdatenbank
ISAy3	ISaGRAF Kommunikationsdaten-Austauschpuffer
ISAx4	ISaGRAF Online-Änderungen 1 Anwendungscode
ISAx5	ISaGRAF Online-Änderungen 2 Anwendungscode
ISAx6	ISaGRAF Anwendungssymbole

Deshalb darf der Benutzer nicht die gleichen Objektnamen benutzen.

- **Anwendungssicherung**

Wenn eine neue Anwendung vom Workstation-Debugger in das Zielsystem geladen wird, wird der Anwendungscode im aktuellen Verzeichnis des Zielsystems unter dem folgenden Dateinamen gesichert:
ISAx1 ISaGRAF Anwendungscode-Sicherungsdatei (x ist die Save-Nummer)

Wenn die Anwendungssymboltabelle vorher geladen wurde, wird sie auch im aktuellen Verzeichnis des Zielsystems gespeichert, und zwar unter dem Namen:
ISAx6 ISaGRAF Anwendungssymbol-Sicherungsdatei (x ist die Slave-Nummer)

Nach dem Start des ISaGRAF Zielsystems werden diese Anwendungscode- und Anwendungssymboldateien im aktuellen Verzeichnis gesucht und unter gleichen Namen als Datenmodule in den Speicher geladen.

Falls keine Symboltabelle im Speicher verfügbar ist, führt das Zielsystem den Anwendungscode ohne geladene Symbole aus.
Falls kein Anwendungscode im Speicher verfügbar ist, wartet das Zielsystem auf das Laden einer Anwendung.

Um das Zielsystem beim Einschalten mit einer bestimmten Anwendung zu starten, ohne daß der Debugger-Anschluß benutzt wird, gibt es folgende Möglichkeiten:

- Die Dateien vom Host-PC, auf dem die Workstation installiert ist, unter Anwendung eines Dateiübertragungswerkzeugs direkt auf die aktuelle Verzeichnisplatte des Zielsystems kopieren. Sie können das Workstation-Werkzeugmenü benutzen (siehe Benutzerhandbuch: Programmverwaltung), um diese Manipulationen zu vereinfachen.

- Den Anwendungscode (und wenn nötig die Anwendungssymboltabelle) aus Dateien vom Host-PC, auf dem die Workstation installiert ist, mit Ihren eigenen Werkzeugen in einem nicht-flüchtigen Speicher (wie PROM oder EPROM) speichern.

  Falls nötig (z.B. weil ein schnellerer Zugiff oder eine schnellere Haltepunktverwaltung gewünscht wird), können Sie den Anwendungscode (und falls nötig die Anwendungssymboltabelle) beim Anlaufen des Systems als **ISAx1** (und falls nötig **ISAx6**) Speicherdatenmodul(e) mit Ihren eigenen Werkzeugen vom PROM zum RAM laden.

  WARNUNG:
  Die Haltepunktverwaltung des ISaGRAF Debuggers kann nicht ordnungsgemäß ausgeführt werden, wenn kein Schreibzugriff zum Anwendungscode-Modul besteht. Dies ist kein Problem, da Ihre Anwendung normalerweise vorher komplett getestet wurde.

Wenn die ISaGRAF Workstation auf dem Host-PC im Standardverzeichnis \ISAWIN installiert ist:
ist die Anwendungscodedatei des Projekts MYPROJ:
\ISAWIN\APL\MYPROJ\appli.x6m (entspricht isax1 auf dem Zielsystem).

ist die Anwendungssymboldatei des Projekts MYPROJ:
\ISAWIN\APL\MYPROJ\appli.tst (entspricht isax6 auf dem Zielsystem).

Fehlerverwaltung und Ausgangsmeldungen

Die ISaGRAF Zielsystem-Software ermöglicht die Verwaltung der Fehlererkennung. Im Anhang finden Sie eine Liste der Fehlermeldungen und deren Beschreibung.

Die Fehlererkennung wird folgendermaßen verarbeitet:
- Ein Fehler setzt sich aus einem Fehler und einer Argumentnummer zusammen, die zur ISaGRAF Fehlerroutine gesendet werden.
- Wenn die Fehlererkennung in den Ausführungsparametern der Workstation aktiviert wurde, wird der Fehler verarbeitet. Ansonsten gehen die Informationen verloren und die Fehlerverwaltung endet.

Bei der Verarbeitung eines Fehlers:
- Fehlernummer (Dezimalwert) und Argument (Hexadezimalwert) werden im Vorgabeausgang stdout angezeigt.
- Fehlernummer und Argument werden in einem FIFO Fehlerpuffer gespeichert, um zu einem späteren Zeitpunkt abgerufen werden zu können. Die Größe des Fehlerpuffers wird in den Ausführungsparametern der Workstation definiert. Wenn der Puffer voll ist, geht bei jedem neu eingehenden Fehler der jeweils älteste Fehler verloren.
- Fehler können entweder vom Debugger oder von der laufenden Anwendung unter Anwendung des SYSTEM-Aufrufs abgerufen werden (siehe Benutzerhandbuch).

Wenn der Debugger einen Fehler erkannt hat, erscheint eine Fehlermeldung mit der Beschreibung des Fehlers im Fehlerfenster. Je nach Anwendungskontext (in Betrieb oder nicht) zeigt der Debugger den Namen des Objekts (Variable oder Programm), in dem der Fehler aufritt, oder den Argumentfehler (Dezimalwert) in Klammern [x] an, was bei jedem Fehler eine unterschiedliche Bedeutung hat.

Eine Willkommensmeldung und die Fehlerwerte werden beim Start des Zielsystems, wenn ein Fehler erkannt wurde, im Vorgabeausgang stdout angezeigt. Wenn diese Anzeige auf dem Standard-Ausgangskanal unerwünscht ist, kann ein Umleitungsbefehl benutzt werden, wie zum Beispiel:

prog_name [options] **>>>/nil**

Zyklusdauer, Task-Verhalten und Task-Prioritäten

- Am Ende eines ISaGRAF Zyklus, unmittelbar bevor ein neuer Zyklus beginnt, wird der folgende Algorithmus ausgeführt:
  Wenn eine Zykluszeit spezifiert ist (von der Workstation: siehe Benutzerhandbuch: Programmverwaltung), dann wird die Zentraleinheit für die verbleibende Zeitspanne (spezifizierte Zykluszeit - aktuelle Zykluszeit der Anwendung) zurückgestellt. Wenn die verbleibende Zeit negativ ist, wird ein Überlauf generiert und die Zentraleinheit um einen Tick zurückgestellt, um die Zeitplanung zu forcieren.

  Wenn keine Zykluszeit spezifiziert wurde oder wenn die verbleibende Zeit weniger oder gleich 1 Tick oder gleich null ist, dann wird die Zentraleinheit um 1 Tick zurückgestellt, um die Zeitplanung zu forcieren.

  Die Zeitpräzision des Zielsystems entspricht einem Tick des OS-9 Systems.

Die spezifizierte Zykluszeit wird gewöhnlich benutzt, um Zyklen auszulösen oder um der Zentraleinheit die Ausführung anderer Tasks auf dem OS-9 System zu gewähren.

- Der Kommunikations-Task befindet sich im "Schlummerzustand", solange keine Daten durch die Kommunikationsverbindung eingehen. Wenn nötig bezieht dieser Task Informationen über die laufende Anwendung mit Hilfe eines Anfrage/Antwort-Protokolls mit dem Kernel-Task. Der Kommunikationstask sendet eine Anfrage an den Kernel. Am Ende des Zyklus (um ein synchrones Anwendungsbild zu erhalten), sendet der Kernel die Antwort an den Kommunikations-Task.

Die ISaGRAF Tasks modifizieren nicht die Prioritäten, die ihnen gegeben wurden. Diese Prioritäten können jedoch, je nach Verhalten der oben beschriebenen ISaGRAF Tasks und den globalen Erfordernissen einer Anwendung, frei vom Benutzer berichtigt werden.
Um sicher zu gehen, daß ISaGRAF nicht von einem Task niedriger Priorität belegt wird, können beispielsweise die OS-9 Taskverwaltungsparameter, wie **MIN_AGE** und **MAX_AGE**, modifiziert werden.

Terminal-Modus

Das serielle Kommunikationsprotokoll des Zielsystems erkennt eine Sequenz von 3 Wagenrücklaufzeichen ($0D) und startet dann einen OS-9 Shell-Task, wenn verfügbar, auf der seriellen Verbindungseinheit.
Dadurch ist es möglich, eine OS-9 Shell-Bedienerführung auf einem beliebigen Terminal unter Anwendung der seriellen ISaGRAF Zielsystem-Verbindung zu erhalten.

Beispiel:
Vom Host-PC:
- Schließen Sie den ISaGRAF Debugger.
- Starten Sie eine Windows-Terminal-Session (Zubehör-Gruppe) mit den richtigen Kommunikationsparametern.
- Schlagen Sie dreimal die Wagenrücklauftaste an.

Sie befinden sich jetzt im OS-9-Shell:
- Tippen Sie **logout**, um den Terminal-Modus zu beenden.

WARNUNG:
Die Terminal-Sitzung darf nur ordnungsgemäß mit dem Befehl logout beendet werden, sonst mislingt der nächste Anschluß an die Workstation.

C.5. Inbetriebnahme des ISaGRAF VxWorks Zielsystems

Für die Ausführung des (der) ISaGRAF Zielsystem(e) ist es zunächst notwendig, einige Befehle auf dem VxWorks System einzuleiten, um die Konfigurationsumgebung einzustellen und schließlich das (die) ISaGRAF Zielsystem(e) zu starten. Diese Befehle können von einer Script-Datei gestartet werden. Sie werden in den folgenden Kapiteln beschrieben.

C.5.1. Der Systemressourcen-Manager: isassr.o

Dieses Modul wird in allen ISaGRAF Zielsystem-Konfigurationen benötigt und muß als erstes der ISaGRAF Zielsystem-Module geladen werden. Es ermöglicht die Verwaltung der Systemressourcen mehrerer gleichzeitig laufender Zielsysteme.

C.5.2. Allgemeine Funktionalitäten von isa.o, isakerse.o, isakeret.o

Um ISaGRAF auszuführen, wird eines der folgenden Module geladen .

isa.o: ermöglicht den Start von ISaGRAF Einzeltask-Zielsystemen (nur mit serieller Kommunikationsverbindung).

Isakerse.o: ermöglicht den Start von ISaGRAF Multitask-Zielsystemen (nur mit serieller Kommunikationsverbindung).

Isakeret.o: ermöglicht den Start von ISaGRAF Multitask-Zielsystemen (mit serieller und/oder Ethernet-Kommunikationsverbindung)

Diese Module werden in den folgenden Kapiteln näher erläutert.

Konfiguration der seriellen Kommunikationsverbindung

Das ISaGRAF Zielsystem benutzt eine serielle Verbindung für die Debugger-Kommunikation. Wenn diese Verbindung vom ISaGRAF Zielsystem aus geöffnet wird, erfolgt die Konfiguration nicht automatisch, sondern kann frei vom Benutzer definiert werden. Hierzu ist allerdings ein binärer Datenübertragungsmodus (RAW Modus) erforderlich. Mit Hilfe der *ISAMOD ()* Subroutine kann eine binäre Verbindung konfiguriert werden:

```
uchar ISAMOD
(
char *desc,                 /* Name der seriellen Einheit */
uint32 baudrate             /* Baud-Geschwindigkeit       */
)
```

Beschreibung:
Konfiguriert die spezifizierte serielle Verbindungseinheit für eine binäre Datenübertragung mit einer bestimmten Baud-Geschwindigkeit.

Rückgabewert:
0 wenn erfolgreich, BAD_RET wenn Fehler auftreten

Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß die Workstation-Kommunikationsparameter (siehe Benutzerhandbuch: Programmverwaltung) mit denen des Zielsystems übereinstimmen.

System-Taktgeberrate

Die globale Variable CLKRATE (uint32) muß mit der VxWorks System-Taktgeberrate initialisiert werden, und zwar folgendermaßen:

CLKRATE = sysClkRateGet ()

Der Vorgabewert von CLKRATE ist 60Hz.

C.5.3. Ausführung des ISaGRAF Einzeltasks: isa.o

Das ISaGRAF Zielsystem kann als Einzeltask ausgeführt werden. Die Operationen in einer solchen Konfiguration können allerdings kritisch sein. Beispielsweise sollte die Kommunikationsverbindung nicht überlastet werden, damit eine gute Leistung garantiert werden kann. Auf dem VxWorks Multitask-System können verschiedene ISaGRAF Einzeltask-Zielsysteme auf einer Zentraleinheit laufen, sofern sie unterschiedliche Slave-Nummern und Kommunikationsanschlüsse besitzen.

Diese Einzeltask-Implementierung wurde hauptsächlich für bestimmte Hardware-Plattformen entworfen, beispielsweise für preisgünstige Karten oder MS-DOS PCs oder um einen ersten Prototyp zu erstellen, wenn er auf eine neue Plattform portiert wird. Daher sollte die Multitasking-Implementierung des ISaGRAF Zielsystems bevorzugt werden.

Das ISaGRAF Einzeltask-Zielsystem verhindert nicht die Ausführung von Hintergrundverarbeitungen oder interruptgesteuerten Routinen.

Slave-Registrierung

Ein ISaGRAF Zielsystem wird durch seine Slave-Nummer gekennzeichnet, die zwischen 1 und 255 liegen kann, mit Ausnahme der Nummer 13 ($0D). Die Slave-Nummer wird im Kommunikationsprotokoll benutzt, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme laufen. Bevor ein oder mehrere ISaGRAF Zielsystem-Tasks gestartet werden, müssen sie registriert werden. Zu diesem Zweck wird die *isa_register_slave()* Subroutine bereitgestellt.

```
uchar isa_register_slave
(
uchar slave                    /* Slave-Nummer */
)
```

Beschreibung:
Fügt dem Verwaltungssystem für Multi-Zielsysteme eine neue Slave-Registrierung hinzu.

Rückgabewert:
0 wenn erfolgreich, BAD_RET wenn Fehler auftreten

Speichereinheit für die Sicherung der Anwendungsdatei

Die globale Variable TSK_FUNIT (char *) kann mit einer Zeichenkette initialisiert werden, die den Pfad der Speichereinheit für die Sicherung der Anwendungsdatei enthält. Das ISaGRAF Zielsystem benutzt lediglich die Standard-Dateiverwaltungs-Routinen fopen, fread, fwrite und fclose für die Sicherung der Anwendungsdatei.

Der Vorgabewert ist eine leere Zeichenkette (""), was bedeutet, daß keine Speichereinheit vorhanden ist.

Beispiel:

TSK_FUNIT = "*host name:*/C:/ISaGRAF/target/apl/"
Spezifiziert ISaGRAF\target\apl\, auf der Wurzel von Einheit C:, auf dem PC *host_name*, als Sicherungsverzeichnis der Anwendungsdatei. Vergessen Sie nicht den letzten Querstrich, sonst erfolgt die Sicherung auf dem Verzeichnis ISaGRAF\target\ mit apl-vorgezeichneten Dateinamen.

Falls notwendig kann diese Variable für jedes Zielsystem und vor jedem Start auf unterschiedliche Speichereinheiten gesetzt werden.
Sie finden ausführliche Informationen über die Sicherung der Anwendungsdateien unter dem Titel "Spezifische Funktionalitäten" im Kapitel Anwendungsicherung.

Steuerung des Zyklusendes

Die Variable TSK_NBTCKSCHED (uint 32) kann auf einen Wert gesetzt werden, der eine Tickverzögerung spezifiziert, die vom ISaGRAF Zielsystem am Ende des Zyklus benutzt wird.
Der Vorgabewert ist 0 (gleiche Taskprioritätsplanung).
Wenn nötig kann diese Variable für jedes Zielsystem und vor jedem Start auf einen anderen Wert gesetzt werden.
Sie finden ausführliche Informationen zu diesem Thema unter "Spezifische Funktionalitäten" in den Kapiteln "Zyklusdauer, Task-Verhalten und Task-Prioritäten".

Starten des ISaGRAF Zielsystems

Nach beendeter Konfiguration wird das ISaGRAF Zielsystem (oder Zielsysteme) gestartet: isa_main.

```
uchar isa_main
(
uchar slave,                    /* Slave-Nummer        */
char *com                       /* Name der seriellen Einheit */
)
```

Beschreibung:
Startet einen ISaGRAF Zielsystem-Task.

Rückgabewert:
Gibt einen Wert ungleich null zurück, wenn ein Fehler auftritt.

Für die Slave-Nummer siehe Kapitel Slave-Registrierung.
Es können mehrere Zielsysteme gestartet werden, sofern sie unterschiedliche Slave-Nummern und Kommunikationsanschlüsse besitzen.
Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß die Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) einem bestehenden Zielsystem entsprechen.

Beispiel

Dieses Beispiel veranschaulicht, wie ein ISaGRAF Einzeltask-Zielsystem mit der Slave-Nummer 1 und mit der seriellen Verbindungseinheit /tyCo/1 gestartet wird.

Das aktuelle Host-Verzeichnis ist jenes, in welchem das Zielsystem installiert ist.

Modul isassr.o laden
ld < RELS/isassr.o

Modul isa.o laden
ld < CMDS/isa.o

Konfiguration der seriellen Kommunikation
ISAMOD ("/tyCo/1", 19200)

Taktgeberrate des Systems
CLKRATE = sysClkRateGet ()

Slave-Registrierung
isa_register_slave (1)

Dateispeichereinheit (kann wegen Vorgabewert ausgelassen werden)
TSK_FUNIT = ""

Steuerung des Zyklusendes (kann wegen Vorgabewert ausgelassen werden)
TSK_NBTCKSCHED = 0

Starten des ISaGRAF Zielsystems
sp (isa_main, 1, "/tyCo/1")

C.5.4. Ausführung der ISaGRAF Multitasks: isakerse.o, isakeret.o

Um die Antwortzeit des ISaGRAF Zielsystem-Kernels und der Kommunikationsverbindung zu verbessern, wird das Zielsystem in zwei Tasks aufgespalten, wodurch die Kommunikationsarbeit (Kommunikations-Task) von der Anwendungsausführung (Kernel-Task) getrennt wird.
Eine solche Architektur ist weitaus flexibler, da sie dem Benutzer ermöglicht, mehr als einen Kommunikations-Task, verbunden mit einem einzigen Kernel-Task, oder bis zu 4 Kernels mit einem einzigen Kommunikations-Task auszuführen. Dies erleichtert bestimmte Integrationen, wie beispielsweise eine Prozeßvisualisierungsverbindung und die Workstation-Debugger-Verbindung auf einer einzigen Anwendung oder eine Einzelverbindung mit 4 verschiedenen Anwendungen durch einen einzigen physischen Anschluß.

Die Kernel- und Kommunikationstasks sind unabhängig voneinander und können getrennt gestartet werden. Es ist lediglich erforderlich, den Kernel-Task (oder Tasks) zuerst zu starten, damit er seine Systemumgebung initialisieren kann und die Kommunikations-Tasks verbunden werden können.

Das ISaGRAF Multitasking-System verhindert nicht die Ausführung von Hintergrundverarbeitungen oder interruptgesteuerten Routinen.

Zwei Module werden angeboten, abhängig von den Kommunikationskapazitäten der Hardware:

- Kernel und serielle Verbindung: isakerse.o

Dieses Modul ermöglicht, den (die) Kernel-Task(s) und den (die) seriellen Kommunikations-Task(s) zu starten.

- Kernel und serielle und/oder Ethernet-Verbindung: isakeret.o
Dieses Modul ermöglicht, den (die) Kernel-Task(s) und den (die) seriellen und/oder Ethernet-Kommunikations-Task(s) zu starten.

Man startet ISaGRAF bei beiden Modulen (isakerse.o und isakeret.o) auf die gleiche Weise, außer daß bei isakeret.o entweder der Name einer seriellen Einheit oder eine Ethernet-Anschlußnummer als Parameter für die Kommunikationseinheit spezifiziert wird, wenn der ISaGRAF Kommunikationstask (oder Tasks) tst_main_ex gestartet wird (siehe unten).

Für ISaGRAF ist das VxWorks-Zielsystem der Server. Der Debugger ist der Client, der sich an die spezifizierte Anschlußnummer anschließt.

Kernel-Registrierung

Ein ISaGRAF Zielsystem wird durch seine Slave-Nummer gekennzeichnet, die zwischen 1 und 255 liegen kann, mit Ausnahme der Nummer 13 ($0D). Diese Slave-Nummer wird im Kommunikationsprotokoll und von dem (den) mit dem Kernel verbundenen Kommunikations-Task(s) benutzt. Sie wird gebraucht, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme laufen. Bevor ein oder mehrere ISaGRAF Zielsystem-Tasks gestartet werden, müssen sie registriert werden. Zu diesem Zweck wird die *isa_register_slave()* Subroutine bereitgestellt.

```
uchar isa_register_slave
  (
  uchar slave    /* Slave-Nummer */
  )
```

Beschreibung:
Fügt dem Verwaltungssystem für Multi-Zielsysteme eine neue Slave-Registrierung hinzu.

Rückgabewert:
0 wenn erfolgreich, BAD_RET wenn Fehler auftreten

Kommunikationstask-Registrierung

Ein ISaGRAF Kommunikationstask wird durch seine logische Nummer gekennzeichnet, die zur gleichzeitigen Verwaltung mehrerer Kommunikationstasks benutzt wird. Sie liegt zwischen 1 und 255 und muß für jeden Kommunikationstasks unterschiedlich sein. Bevor Sie den ISaGRAF Kommunikationstask (oder Tasks) starten, muß dieser registriert werden. Hierzu wird die Subroutine *isa_register_com()* benutzt.

```
uchar isa_register_com
(
uchar com_id                    /* Kommunikationstask-Bezeichner */
)
```

Beschreibung:
Fügt dem Verwaltungssystem für Multi-Zielsysteme eine neue Kommunikationstask-Registrierung hinzu.

Rückgabewert:
0 wenn erfolgreich, BAD_RET wenn Fehler auftreten

Speichereinheit für die Sicherung der Anwendungsdatei

Die globale Variable TSK_FUNIT (char *) kann mit einer Zeichenkette initialisiert werden, die den Pfad der Speichereinheit für die Sicherung der Anwendungsdatei enthält. Das ISaGRAF Zielsystem benutzt lediglich die Standard-Dateiverwaltungs-Routinen fopen, fread, fwrite und fclose für die Sicherung der Anwendungsdatei.
Der Vorgabewert ist eine leere Zeichenkette (""), was bedeutet, daß keine Speichereinheit vorhanden ist.

Beispiel:
TSK_FUNIT = "*host name:*/C:/ISaGRAF/target/apl/"
Spezifiziert ISaGRAF\target\apl\, auf der Wurzel von Einheit C:, auf dem PC *host_name*, als Sicherungsverzeichnis der Anwendungsdatei. Vergessen Sie nicht den letzten Querstrich, sonst erfolgt die Sicherung auf dem Verzeichnis ISaGRAF\target\ mit apl-vorgezeichneten Dateinamen.

Falls notwendig kann diese Variable für jedes Zielgerät und vor jedem Start auf unterschiedliche Speichereinheiten gesetzt werden.
Sie finden ausführliche Informationen über die Sicherung der Anwendungsdateien unter dem Titel "Spezifische Funktionalitäten" im Kapitel "Anwendungsicherung".

Steuerung des Zyklusendes

Die Variable TSK_NBTCKSCHED (uint 32) kann auf einen Wert gesetzt werden, der eine Tickverzögerung spezifiziert, die vom ISaGRAF Zielsystem am Ende des Zyklus benutzt wird.
Der Vorgabewert ist 0 (gleiche Taskprioritätsplanung).
Wenn nötig kann diese Variable für jedes Zielsystem und vor jedem Start auf einen anderen Wert gesetzt werden.
Sie finden ausführliche Informationen zu diesem Thema unter "Spezifische Funktionalitäten" in den Kapiteln "Zyklusdauer, Task-Verhalten und Taks-Prioritäten".

Starten des ISaGRAF Kernels

Nach beendeter Konfiguration wird der ISaGRAF Kernel (oder Kernels) gestartet: isa_main.

```
uchar isa_main
(
uchar slave,                    /* Slave-Nummer        */
char *com                       /* NOT USED Leere Zeichenkette ist OK */
)
```

Beschreibung:
Startet einen ISaGRAF Kerneltask

Rückgabewert:
Rückgabewert ungleich null, wenn ein Fehler auftritt.

Für die Slave-Nummer siehe Kapitel Slave-Registrierung.

Es können mehrere Kernels gestartet werden, sofern sie unterschiedliche Slave-Nummern besitzen.

Starten des ISaGRAF Kommunikationstasks

Nach beendeter Konfiguration wird der ISaGRAF Kommunikationstask (oder Tasks) gestartet: tst_main_ex.

```
uchar tst_main_ex
(
char *com,                    /* Name der Kommunikationseinheit */
uchar *slave,                 /* Lokalisierung eines 4 Byte-Felds, das den
Kernel-Slave für die  Verbindung spezifiziert */
uchar com_id                  /* Kommunikationstask-Bezeichner */
)
```

Beschreibung:
Startet einen ISaGRAF Kommunikationstask.

Rückgabewert:
Rückgabewert ist ungleich null, wenn ein Fehler auftritt.

Das 4 Byte-Feld spezifiziert den (die) Kernel-Slave(s), mit dem (denen) der Kommunikationstask verbunden ist. Wenn weniger als 4 Kernel-Slaves gebraucht werden, muß das Feld mit Nullen vervollständigt werden. Wenn der Task angelaufen ist, wird das Feld nicht mehr gebraucht.
Der Name der Kommunikationseinheit entspricht dem Namen der seriellen Einheit, der für die Kommunikationsverbindung benutzt wird.
Es können mehrere Kommunikationstasks gestartet werden, sofern sie unterschiedliche Task-Bezeichner besitzen.
Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß die Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) einem bestehenden Zielsystem entsprechen (Kernel- und Kommunikationstasks).

Beispiel:

Im nachfolgenden Beispiel werden die folgenden Tasks gestartet:
Ein ISaGRAF Kerneltask mit der Slave-Nummer 1.
Ein ISaGRAF Kommunikationstask gekennzeichnet durch Nummer 1, verbunden mit dem Kernel-Slave 1 und mit der Einheit /tyCo/1 für die serielle Verbindung.
Ein ISaGRAF Kommunikationstask gekennzeichnet durch Nummer 2, verbunden mit dem Kernel-Slave 1 und mit der Anschlußnummer 1100 für die Ethernet-Kommunikationsverbindung.
Das aktuelle Host-Verzeichnis ist jenes, in welchem das Zielsystem installiert ist.

Modul isassr.o laden
ld < RELS/isassr.o

Modul isakeret.o laden (Sie können auch isakerse.o laden, wenn keine Ethernet-Kommunikationsverbindung benötigt wird)
ld < CMDS/isakeret.o

Konfiguration der seriellen Kommunikation
ISAMOD ("/tyCo/1", 19200)

System-Taktgeberrate
CLKRATE = sysClkRateGet ()

Slave-Registrierung
isa_register_slave (1)

Kommunikationsregistrierung
isa_register_com (1)
isa_register_com (2)

Dateispeichereinheit (kann wegen Vorgabewert ausgelassen werden)
TSK_FUNIT = ""

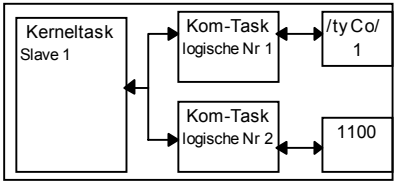
Zyklusendsteuerung (kann wegen Vorgabewert ausgelassen werden)
TSK_NBTCKSCHED = 0

Starten des ISaGRAF Kernels
sp (isa_main, 1, "")

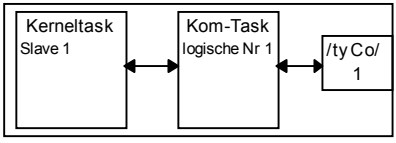
Kommunikationstask, Slave-Verbindung
SlavesLink = 0x01000000

Starten der ISaGRAF Kommunikationstasks
sp (tst_main_ex, "/tyCo/1", &SlavesLink, 1)
sp (tst_main_ex, "1100", &SlavesLink, 2)

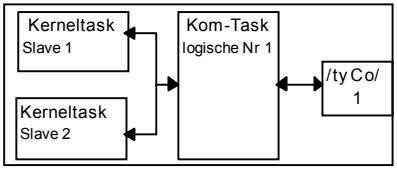
Dieser Start entspricht dem folgenden Schema:



Sie können auch eine der folgenden Basiskonfigurationen wählen:



Die einfachste Konfiguration besteht aus einem Kerneltask, der einem Kommunikationstask auf einer seriellen (Ethernet-) Verbindung zugeordnet ist.

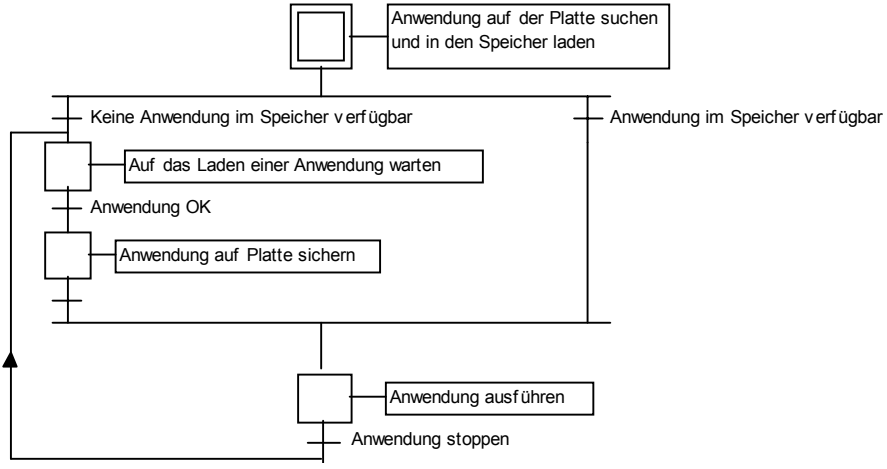


Eine andere Konfiguration besteht aus 2 Kernels, die einem Kommunikationstask auf einer seriellen (Ethernet-) Verbindung zugeordnet sind. In diesem Fall, SlavesLink = 0x01020000.

C.5.5. Spezifische Funktionalitäten

Starten von ISaGRAF

Nach dem Start des Zielsystems wird der folgende Algorithmus ausgeführt.



- **Definitionen**

Der Anwendungscode ist die binäre Datenbank, welche von der Workstation generiert und geladen und dann vom Zielsystem ausgeführt wird. Sie kann durch die Symboltabelle ergänzt werden.

Die Symboltabelle einer Anwendung ist eine ASCII-Datenbank, die von der Workstation generiert und geladen wird. Diese Tabelle stellt die Verbindung zwischen Symbolobjekten und internen Zielsystemobjekten her. Sie ist im Zielsystem nicht erforderlich, außer für die Verwaltung von benutzerspezifischen Symbolen. Für weitere Informationen über die Symboltabellen siehe Benutzerhandbuch: Fortgeschrittene Programmierungstechniken.

Der Pfad zur Plattendateieinheit wird beim Start des ISaGRAF Zielsystems mit Hilfe der globalen Variablen TSK_FUNIT spezifiziert (Vorgabewert = "" bedeutet, daß keine Plattendateieinheit vorhanden ist).

- **ISaGRAF Multi-Anwendungen**

Verschiedene Anwendungen (Kernel- und Kommunikationstasks) können auf einer Zentraleinheit gleichzeitig laufen, sofern sie jeweils unterschiedliche Slave-Nummern und logische Kommunikationstask-Nummern haben. Bei der Ausführung verschiedener Anwendungen sollte der Benutzer jedoch bei geteiltem Zugriff bestimmter Anwendungsobjekte, wie E/A-Karten, äußerst vorsichtig handeln. Verschiedene Anwendungen (Kernels) können beispielsweise getrennte physische Karten benutzen, außer es wird eine Art E/A-Server oder Semaphor durch den E/A-Treiber implementiert.

- **Anwendungssicherung**

Wenn eine neue Anwendung vom Workstation-Debugger in das Zielsystem geladen wird, wird der Anwendungscode (das Zielsystem benutzt die Standard-Dateiverwaltungs-Routinen fopen,...) unter dem folgenden Dateinamen gesichert:

*path***ISAx1** ISaGRAF Anwendungscode-Sicherungsdatei (x ist die Slave-Nummer)

Wenn die Anwendungssymboltabelle vorher geladen wurde, wird sie auch im aktuellen Verzeichnis des Zielsystems gespeichert, und zwar unter dem Namen:

*path***ISAx6** ISaGRAF Anwendungssymbol-Sicherungsdatei (x ist die Slave-Nummer)

Der *path* wird beim Start des ISaGRAF Zielsystem mit Hilfe der globalen Variablen TSK_FUNIT spezifiziert. Eine leere Zeichenkette ("") bedeutet, daß keine Plattendateieinheit vorhanden ist (Vorgabewert).

Nach dem Start des ISaGRAF Zielsystems werden diese Anwendungscode- und Anwendungssymboldateien im aktuellen Verzeichnis gesucht und unter gleichen Namen als Datenmodule in den Speicher geladen.

Falls keine Symboltabelle im Speicher verfügbar ist, führt das Zielsystem den Anwendungscode ohne geladene Symbole aus.
Falls kein Anwendungscode im Speicher verfügbar ist, wartet das Zielsystem auf das Laden einer Anwendung.

Um das Zielsystem beim Einschalten mit einer bestimmten Anwendung zu starten, ohne daß der Debugger-Anschluß benutzt wird, gibt es folgende Möglichkeiten:

- Die Dateien vom Host-PC, auf dem die Workstation installiert ist, unter Anwendung eines Dateiübertragungswerkzeugs direkt auf die aktuelle Verzeichnisplatte des Zielsystems kopieren. Sie können das Workstation-Werkzeugmenü benutzen (siehe Benutzerhandbuch: Programmverwaltung), um diese Manipulationen zu vereinfachen.

- Den Anwendungscode (und wenn nötig die Anwendungssymboltabelle) aus Dateien vom Host-PC, auf dem die Workstation installiert ist, mit Ihren eigenen Werkzeugen in einem nicht-flüchtigen Speicher (wie PROM oder EPROM) speichern.

  Falls nötig (z.B. weil ein schnellerer Zugiff oder eine schnellere Haltepunktverwaltung gewünscht wird), können Sie den Anwendungscode (und falls nötig die Anwendungssymboltabelle) beim Anlaufen des Systems mit Ihren eigenen Werkzeugen vom PROM zum RAM laden.

  Beim Start von ISaGRAF (bevor die Tasks eingeleitet werden) müssen Sie die Adresse(n) spezifizieren, an der (denen) der Anwendungscode (und falls nötig, die Anwendungssymboltabelle) gespeichert ist. Dazu wird die globale Variable SSR wie folgt initialisiert:

SSR[x][1].space = *Adresse des Anwendungscodes*
und falls nötig:
SSR[x][6].space = *Adresse der Anwendungssymboltabelle*

Zu diesem Zweck können Sie eine kurze Prozedur schreiben. Die globale Variable SSR wird als ein str_ssr Strukturtyp deklariert, der in der Datei tasy0ssr.h definiert wird.

WARNUNG:
Die Haltepunktverwaltung des ISaGRAF Debuggers kann nicht ordnungsgemäß ausgeführt werden, wenn kein Schreibzugriff zum Anwendungscode besteht. Dies ist kein Problem, da Ihre Anwendung normalerweise vorher komplett getestet wurde.

Wenn die ISaGRAF Workstation auf dem Host-PC im Standardverzeichnis \ISAWIN installiert ist:
ist die Anwendungscodedatei des Projekts MYPROJ:
\ISAWIN\APL\MYPROJ\appli.x6m (entspricht isax1 auf dem Zielsystem).
ist die Anwendungssymboldatei des Projekts MYPROJ:
\ISAWIN\APL\MYPROJ\appli.tst (entspricht isax6 auf dem Zielsystem).

Fehlerverwaltung und Ausgangsmeldungen

Die ISaGRAF Zielsystem-Software integriert die Verwaltung der Fehlererkennung. Im Anhang finden Sie eine Liste der Fehlermeldungen und deren Beschreibung.

Die Fehlererkennung wird folgendermaßen verarbeitet:
- Ein Fehler setzt sich aus einem Fehler und einer Argumentnummer zusammen, die zur ISaGRAF Fehlerroutine gesendet werden.
- Wenn die Fehlererkennung in den Ausführungsparametern der Workstation aktiviert wurde, wird der Fehler verarbeitet. Ansonsten gehen die Informationen verloren und die Fehlerverwaltung endet.

Bei der Verarbeitung eines Fehlers:
- Fehlernummer (Dezimalwert) und Argument (Hexadezimalwert) werden im Vorgabeausgang stdout angezeigt.
- Fehlernummer und Argument werden in einem FIFO Fehlerpuffer gespeichert, um zu einem späteren Zeitpunkt abgerufen werden zu können. Die Größe des Fehlerpuffers wird in den Ausführungsparametern der Workstation definiert. Wenn der Puffer voll ist, geht bei jedem neu eingehenden Fehler der jeweils älteste Fehler verloren.
- Fehler können entweder vom Debugger oder von der laufenden Anwendung unter Anwendung des SYSTEM-Aufrufs abgerufen werden (siehe Benutzerhandbuch).

Wenn der Debugger einen Fehler erkannt hat, erscheint eine Fehlermeldung mit der Beschreibung des Fehlers im Fehlerfenster. Je nach Anwendungskontext (in Betrieb oder nicht) zeigt der Debugger den Namen des Objekts (Variable oder Programm), in dem der Fehler aufritt, oder den Argumentfehler (Dezimalwert) in Klammern [x] an, was bei jedem Fehler eine unterschiedliche Bedeutung hat.
Wenn ein Fehler erkannt wurde, werden die Fehlerwerte auf dem Zielsystem im Vorgabeausgang stdout angezeigt. Diese Anzeige kann mit Hilfe von VxWorks-Routinen geleitet werden, wie zum Beispiel:
ioGlobalStdSet() oder *ioTaskStdSet()*

Im zweiten Fall, damit weder der Kernel noch die Kommunikationstasks Fehler generieren können.

Zyklusdauer, Task-Verhalten und Task-Prioritäten

Am Ende eines ISaGRAF Zyklus, unmittelbar bevor ein neuer Zyklus beginnt, wird der folgende Algorithmus ausgeführt:
Wenn eine Zykluszeit spezifiziert wurde (von der Workstation: siehe Benutzerhandbuch: Programmverwaltung) wird die Zentraleinheit für die verbleibende Zeitperiode zurückgestellt (spezifizierte Zykluszeit - aktuelle Zykluszeit der Anwendung). Wenn die verbleibende Zeitperiode negativ ist, wird ein Überlauf generiert und die Zentraleinheit um TSK_NBTCKSCHED (beim Start von ISaGRAF eingestellt Variable) Tick(s) zurückgestellt, um die Zeitplanung zu forcieren.

Wenn keine Zykluszeit spezifiziert wurde oder wenn die verbleibende Zeit weniger als 1 Tick oder gleich null ist, dann wird die Zentraleinheit um TSK_NBTCKSCHED Tick(s) zurückgestellt, um die Zeitplanung zu forcieren.

Die Genauigkeit des Zielsystems entspricht einem VxWorks-Systemtick.

Die spezifizierte Zykluszeit wird gewöhnlich benutzt, um Zyklen auszulösen oder um die Zentraleinheit anderen Tasks, welche auf dem VxWorks-System laufen, zu überlassen.

- Der Kommunikationstask befindet sich im "Schlummerzustand", solange keine Daten durch die Kommunikationsverbindung eingehen. Wenn nötig bezieht dieser Task Informationen über die laufende Anwendung durch ein Anfrage/Antwortprotokoll mit dem Kerneltask. Der Kommunikationstask sendet eine Anfrage an den Kernel. Am Ende des Zyklus (um ein synchrones Anwendungsbild zu erhalten) beantwortet der Kernel die Anfrage des Kommunikationstasks.

Die ISaGRAF Tasks modifizieren nicht die Prioritäten, die ihnen gegeben wurden. Diese Prioritäten können jedoch, je nach Verhalten der oben beschriebenen ISaGRAF Tasks und den globalen Erfordernissen einer Anwendung, frei vom Benutzer berichtigt werden.

C.6. Inbetriebnahme des ISaGRAF NT Zielsystems

C.6.1. Ausführung von ISaGRAF

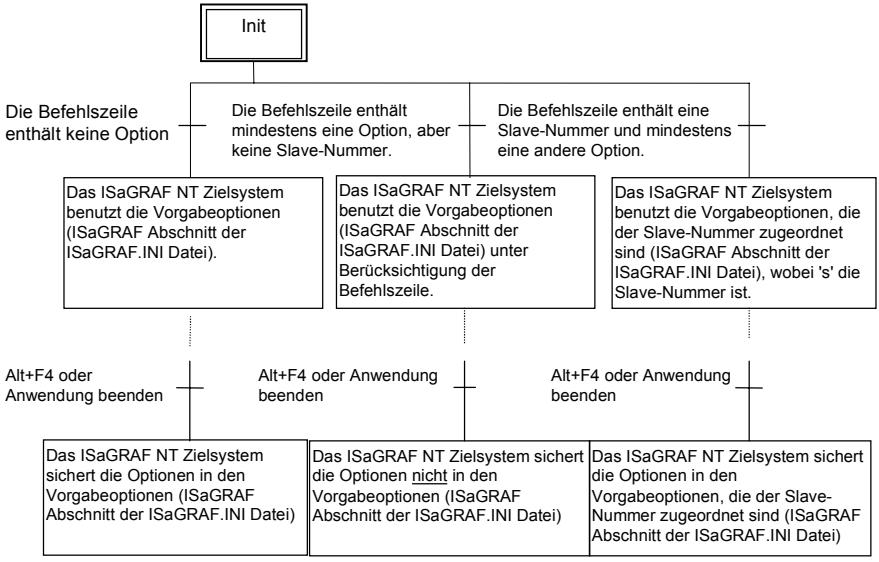
In der NT-Implementierung läuft das Zielsystem als ein einziges Programm: WISAKER.EXE, das mehrmals gestartet werden kann. Auf diese Weise können beliebig viele ISaGRAF NT Zielsysteme betrieben werden, da jede Instance ihre eigene Slave-Nummer hat.

Das Zielsystemprogramm verhindert nicht die Ausführung von interruptgesteuerten Routinen.

Die WISAKER-Software wurde für die Ausführung unter Windows NT 3.51 oder höher entwickelt.

C.6.2. Allgemeine Informationen zu den Optionen

Die Optionen werden entsprechend des folgenden Diagramms gesichert und abgerufen:



Hinweis: Die ISAGRAF.INI Datei wird im aktuellen Verzeichnis gesichert.

- ***Slave-Nummer: -s Option***

Diese Option definiert die Slave-Nummer des Zielsystems. Sie kann zwischen 1 und 255 liegen, mit Ausnahme der Nummer 13 ($0D). Die Slave-Nummer wird im Kommunikationsprotokoll benutzt, um zwischen Slaves zu unterscheiden, wenn mehrere Zielsysteme mit der Host-Workstation verbunden sind oder wenn mehrere Zielsysteme auf einem PC laufen. Wenn Sie den Workstation-Debugger benutzen, vergewissern Sie sich, daß der Slave-Parameter der Workstation (siehe Benutzerhandbuch: Programmverwaltung) mit dem des Zielsystems übereinstimmt.

Vorgabewert: Die vorgegebene Slave-Nummer ist 1 oder die der ISaGRAF.INI Datei.

Beispiel:
WISAKER.EXE -s=2

Benutzerschnittstelle: Dieses Fenster wird mit dem Befehl "Optionen/Slave" des Hauptfensters des ISaGRAF NT Zielsystems angezeigt.

Mit Hilfe der Maus oder der Pfeilschaltflächen (Nach oben und Nach unten) kann der Wert dieser Option verändert werden. Um den neuen Wert benutzen zu können, muß das ISaGRAF NT Zielsystem erneut gestartet werden.

Kommunikationsverbindung und Konfiguration: -t Option

Das ISaGRAF Zielsystem kann eine serielle oder eine Ethernet-Verbindung für die Kommunikation mit dem Debugger benutzen.
Der Name des Anschlusses wird mit der -t Option spezifiziert. Da die Kommunikationsschnittstelle so konzipiert wurde, daß sie mit einem beliebigen Gerät kompatibel ist, können die Anschlüsse COM1, COM2, COM3 oder COM4 für die serielle Kommunikation und Anschlußnummern ab 1100 für die Ethernet-Kommunikation benutzt werden.

Vorgabewert: Der vorgegebene Kommunikationsanschluß ist 1100 für Ethernet und COM1 für die serielle Kommunikation (oder der der ISaGRAF.INI Datei).

HINWEIS: Als Kommunikationsverbindung wird Ethernet vorgegeben.

Beispiele:
WISAKER -t=COM2
WISAKER -t=1101

Serielle Konfiguration:
Manche Optionen können nur benutzt werden, wenn die -t=COMx Option spezifiziert ist.

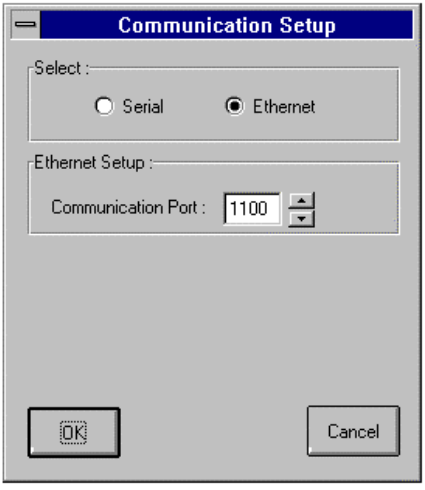
Diese Optionen werden für die Konfiguration der seriellen Verbindung benutzt:

Option	Werte	Bedeutung
baud	600 1200 2400 4800 9600 **19200**	Baud-Geschwindigkeit
parity	**n** e o	Keine Parität Gerade Ungerade
data	7 o. **8**	Anzahl der Bits
stop	**1** o. 2	Länge des Stoppbits
flow	h **n**	Hardware-Steuerung Keine Steuerung

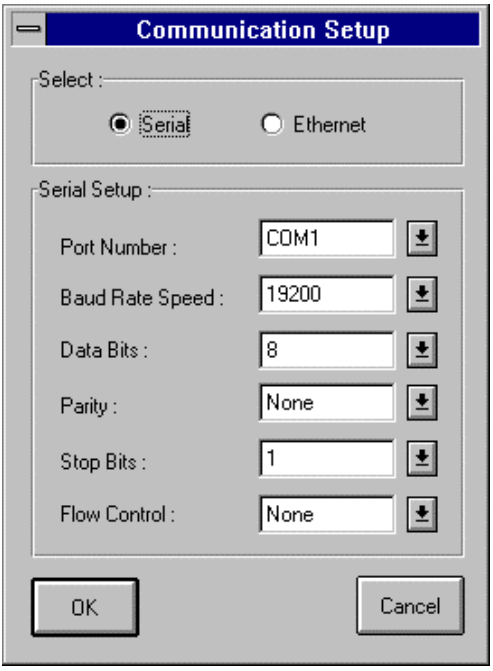
Die Vorgabewerte sind 19200, keine Parität, 8 Datenbits, 1 Stoppbit, keine Steuerung

Beispiel:
WISAKER -t=COM1 baud=1200 data=8 parity=n stop=2

Benutzerschnittstelle: Dieses Fenster wird mit dem Befehl "Optionen/Kommunikation" des Hauptfensters des ISaGRAF NT Zielsystems angezeigt.



Sie haben die Wahl zwischen der seriellen Kommunikation und der Ethernet-Kommunikation. Wenn Sie Ethernet wählen, können Sie die Anschlußnummer ändern. Die gewählte Anschlußnummer sollte die gleiche sein, wie die in der Workstation für die PC-SPS Verbindung spezifizierte Nummer.



Wenn Sie die serielle Kommunikation wählen, erscheint die Konfiguration. Diese Konfiguration sollte die gleiche sein, wie die der PC-SPS Verbindung der Workstation.

Grafische Simulierung virtueller Karten: -x Option

Wenn diese Option gesetzt wird, werden die im E/A-Verdrahtungseditor (siehe Teil A) als virtuell deklarierten Karten simuliert.
Mögliche Werte sind 0 oder 1, (0 bedeutet keine Simulation, 1 bedeutet Simulation aktiviert).

Vorgabewert: Der Vorgabewert ist 0 oder der Wert der ISaGRAF.INI Datei.

Beispiel:
WISAKER -x=1 simuliert virtuelle Karten,

Benutzerschnittstelle: Das Menüelement erscheint angekreuzt oder nicht, je nach Status der Option. Die simulierten Karten erscheinen in einer grafischen Tafel.

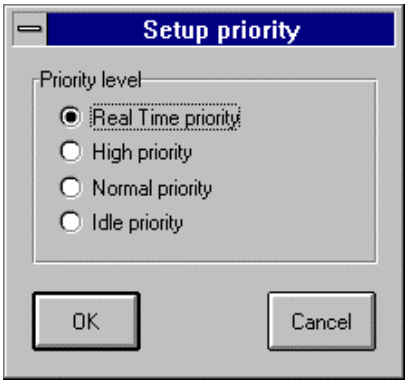
Priorität des ISaGRAF NT Zielsystems: -p Option

Da das Zielsystem unter NT betrieben wird, ist es äußerst nützlich, eine Prioritätsstufe zu definieren. Beispielsweise ist es möglich, eine zeitkritische ISaGRAF Anwendung innerhalb eines Zielsystems mit einer höheren Priorität auszuführen oder ein oder mehrere Zielsysteme mit niedrigeren Prioritäten im Hintergrund auszuführen.

Mögliche Werte sind 0, 1, 2 oder 3. (0 ist die höchste Priorität und 3 die niedrigste Priorität).

Beispiele:
WISAKER -p=0
WISAKER -p=1

Benutzerschnittstelle: Dieses Fenster wird mit dem Befehl "Optionen/Priorität" im Hauptfenster des ISaGRAF NT-Zielsystems angezeigt.



Die höchste Priorität ist Echtzeit und die niedrigste Priorität Leerlauf.
0: Echtzeitpriorität
1: Hohe Priorität
2: Normale Priorität
3: Leerlaufpriorität

Beispiele:

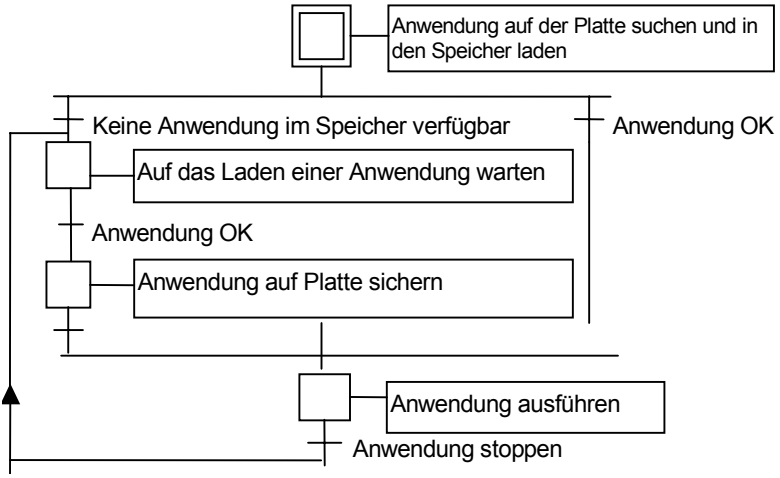
wisaker -t=COM1 Startet das ISaGRAF Zielsystem mit Vorgabe-Slave-Nummer (1) und mit COM1 als Kommunikationsanschluß.

wisaker -s=3 -t=COM1 Startet das ISaGRAF Zielsystem mit Slave-Nummer 3 und mit COM1 als Kommunikationsanschluß.

C.6.3. Spezifische Funktionalitäten

Starten von ISaGRAF

Beim Start des Zielsystems wird der folgende Algorithmus ausgeführt.



- **Definitionen**

Der Anwendungscode ist die binäre Datenbank, welche von der Workstation generiert und geladen und dann vom Zielsystem ausgeführt wird. Sie kann durch die Symboltabelle ergänzt werden.
Die Symboltabelle einer Anwendung ist eine ASCII-Datenbank, die von der Workstation generiert und geladen wird. Diese Tabelle stellt die Verbindung zwischen Symbolobjekten und internen Zielsystemobjekten her. Sie ist im Zielsystem nicht erforderlich, außer für die Verwaltung von benutzerspezifischen Symbolen, wie zum Beispiel die DDE-Funktion oder die E/A-Simulation mit Symbolnamen. Für weitere Informationen über Symboltabellen siehe Benutzerhandbuch: Fortgeschrittene Programmierungstechniken.

- **ISaGRAF Multi-Anwendungen**

Verschiedene Anwendungen können auf einer Zentraleinheit gleichzeitig laufen, sofern sie jeweils unterschiedliche Slave-Nummern und logische Kommunikationstask-Nummern haben. Bei der Ausführung verschiedener Anwendungen sollte der Benutzer jedoch bei geteiltem Zugriff bestimmter Anwendungsobjekte, wie E/A-Karten, äußerst vorsichtig handeln. Verschiedene Anwendungen können beispielsweise getrennte physische Karten benutzen, außer es wird eine Art E/A-Server oder Semaphor durch den E/A-Treiber implementiert.

- **Anwendungssicherung**

Wenn eine neue Anwendung vom Workstation-Debugger in das Zielsystem geladen wird, wird der Anwendungscode im aktuellen Verzeichnis des Zielsystems unter dem folgenden Dateinamen gesichert:

ISAx1 ISaGRAF Anwendungscode-Sicherungsdatei (x ist die Slave-Nummer)

Wenn die Anwendungssymboltabelle vorher geladen wurde, wird sie auch im aktuellen Verzeichnis des Zielsystems gespeichert, und zwar unter dem Namen:

ISAx6 ISaGRAF Anwendungssymbol-Sicherungsdatei (x ist die Slave-Nummer)

Nach dem Start des ISaGRAF Zielsystems werden diese Anwendungscode- und Anwendungssymboldateien im aktuellen Verzeichnis gesucht und in den Speicher geladen.

Falls keine Symboltabelle im Speicher verfügbar ist, führt das Zielsystem den Anwendungscode ohne geladene Symbole aus.
Falls kein Anwendungscode im Speicher verfügbar ist, wartet das Zielsystem auf das Laden einer Anwendung.

Um das Zielsystem beim Einschalten mit einer bestimmten Anwendung zu starten, ohne daß der Debugger-Anschluß benutzt wird, kopiert man diese Dateien von der Platte (wenn sich die Workstation auf demselben PC befindet) oder unter Anwendung einer Diskette direkt in das aktuelle Verzeichnis des Zielsystems.

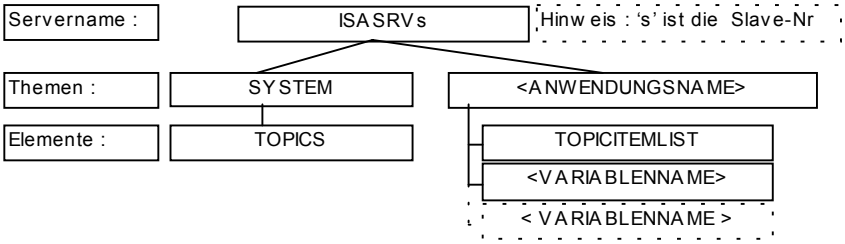
Wenn die ISaGRAF Workstation im Standardverzeichnis \ISAWIN installiert ist,
ist die Anwendungscodedatei des Projekts MYPROJ:
\ISAWIN\APL\MYPROJ\appli.x8m
und die Anwendungssymboldatei des Projekts MYPROJ:
\ISAWIN\APL\MYPROJ\appli.tst

Beispiel:
Vom Verzeichnis, in dem WISAKER.EXE installiert ist, wenn der folgende Befehl eingegeben wird:
copy \ISAWIN\APL\MYPROJ\appli.x8m isa11
Dann findet WISAKER.EXE die Anwendung 'myproj' und führt sie aus.

Alle diese Befehle können gruppiert werden, beispielsweise in einer Batch-Datei, und dann vom Workstation-Werkzeugmenü aus gestartet werden (siehe Benutzerhandbuch: Programmverwaltung)

DDE-Spezifizierung

Das ISaGRAF NT Zielsystem ist ein DDE-Server (Dynamic Data Exchange). Eine beliebige Client-Software kann mit dem Zielsystem verbunden werden, um Variablen auszutauschen. Zum Beispiel kann MSEXCEL Grafiken mit Werten animieren, die vom ISaGRAF Zielsystem über DDE eingehen.
Die DDE-Funktion benötigt die Anwendungssymboltabelle auf dem Zielsystem.
DDE-Subjekte werden wie folgt definiert:



« ISASRVs » ist der Name des DDE-Servers, 's' ist die Slave-Nummer.
« SYSTEM » ist ein Standardthema, das Zugriff zum « TOPICS » Element gibt,

« TOPICS » liefert die Liste der definierten Themen: System und den Namen der Anwendung, die im ISaGRAF NT Zielsystem läuft.
« ANWENDUNGSNAME » ist der Name der Anwendung.
« TOPICITEMLIST » ist die Liste der verfügbaren Elemente unter dem aktuellen Thema, dies liefert die Liste der Variablen, auf die über DDE zugegriffen werden kann.
« VARIABLENNAME » ist der Name einer Variablen.

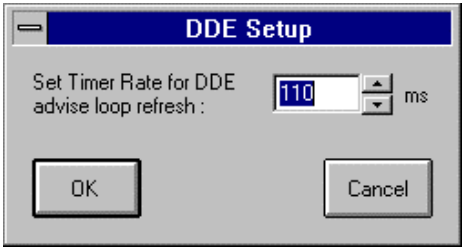
DDE Advise Loop Rate für ISaGRAF NT Zielsysteme: -d Option
Im allgemeinen ruft der DDE-Client die Variablen jedesmal dann auf, wenn sie gebraucht werden. Wenn die Variablen einer Anwendung sehr zahlreich sind, kann dieser Vorgang sehr viel Zeit in Anspruch nehmen. Es gibt einen Modus, der Advise-Modus (advise loop) genannt wird, in dem der Server nur die modifizierten Variablen sendet, so daß die Kommunikation reduziert werden kann. In diesem Modus werden die als "advised" markierten Variablen in regelmäßigen Abständen vom Server überwacht, um zu entscheiden, welche Variablen gesendet werden sollen. Diese Periode wird DDE Advise Loop Rate genannt.

Die DDE Advise Loop Rate (in ms) wird mit der -d Option spezifiziert.

Vorgabewert: Der Vorgabewert ist 1000 ms oder der in der ISaGRAF.INI Datei spezifizierte Wert.

Beispiel:
WISAKER -d=100

Benutzerschnittstelle: Dieses Fenster wird mit dem Befehl "Optionen/DDE" im Hauptfenster des ISaGRAF NT Zielsystems angezeigt.



Fehlerverwaltung und Ausgangsmeldungen

Die ISaGRAF Zielsystem-Software ermöglicht die Verwaltung der Fehlererkennung. Im Anhang finden Sie eine Liste der Fehlermeldungen und deren Beschreibung.

Die Fehlererkennung wird folgendermaßen verarbeitet:
- Ein Fehler setzt sich aus einem Fehler und einer Argumentnummer zusammen, die zur ISaGRAF Fehlerroutine gesendet werden.
- Wenn die Fehlererkennung in den Ausführungsparametern der Workstation aktiviert wurde, wird der Fehler verarbeitet. Ansonsten gehen die Informationen verloren und die Fehlerverwaltung endet.

Bei der Verarbeitung eines Fehlers:

- Fehlernummer (Dezimalwert) und Argument (Hexadezimalwert) werden im Ausgang angezeigt (Fenster von WISAKER.EXE).
- Fehlernummer und Argument werden in einem FIFO Fehlerpuffer gespeichert, um zu einem späteren Zeitpunkt abgerufen werden zu können. Die Größe des Fehlerpuffers wird in den Ausführungsparametern der Workstation definiert. Wenn der Puffer voll ist, geht bei jedem neu eingehenden Fehler der jeweils älteste Fehler verloren.
- Fehler können entweder vom Debugger oder von der laufenden Anwendung unter Anwendung des SYSTEM-Aufrufs abgerufen werden (siehe Benutzerhandbuch).

Wenn der Debugger einen Fehler erkannt hat, erscheint eine Fehlermeldung mit der Beschreibung des Fehlers im Fehlerfenster. Je nach Anwendungskontext (in Betrieb oder nicht) zeigt der Debugger den Namen des Objekts (Variable oder Programm), in dem der Fehler auftritt, oder den Argumentfehler (Dezimalwert) in Klammern [x] an, was bei jedem Fehler eine unterschiedliche Bedeutung hat.

Eine Willkommensmeldung wird beim Start des Zielsystems im Ausgang angezeigt. Sie setzt sich aus der Slave-Nummer, der Kommunikationskonfiguration und dem DDE-Servernamen zusammen.

Systemuhr

Da das ISaGRAF NT Zielsystem so entworfen wurde, daß es auf beliebigen Systemen laufen kann, wird der Standard-Tick als Zeitreferenz für die Zyklussynchronisierung und die Aktualisierung der Timer-Variablen benutzt. Er beträgt 10 Millisekunden.

Daher ist es nicht möglich, eine Genauigkeit der Timer-Variablen von mehr als 10 ms zu erzielen. Aus diesem Grund generiert eine spezifische Zykluszeit weniger oder gleich 10 ms und ungleich null einen Zykluszeitüberlauffehler (Fehler 62). Siehe folgende Kapitel für weitere Informationen.

Fragen Sie Ihren Lieferanten nach einer spezifischen Implementierung, wenn Ihre Anwendung eine größere Genauigkeit erfordert.

Zyklusdauer und Zielsystemverhalten

Am Ende eines ISaGRAF Zyklus, unmittelbar bevor ein neuer Zyklus beginnt, wird der folgende Algorithmus ausgeführt:

Wenn eine Zykluszeit spezifiziert wurde (von der Workstation: siehe Benutzerhandbuch: Programmverwaltung) wird die Zentraleinheit für die verbleibende Zeitperiode zurückgestellt (spezifizierte Zykluszeit - aktuelle Zykluszeit der Anwendung). Wenn die verbleibende Zeitperiode negativ ist, wird ein Überlauf generiert und die Zentraleinheit um 1 Tick zurückgestellt, um die Zeitplanung zu forcieren.

Wenn keine Zykluszeit spezifiziert wurde oder wenn die verbleibende Zeit weniger als 1 Tick oder gleich null ist, dann wird die Zentraleinheit um 1 Tick zurückgestellt, um die Zeitplanung zu forcieren

Die Zeitgenauigkeit des Zielsystems entspricht einem Tick des Windows NT Systems.

Die spezifizierte Zykluszeit wird gewöhnlich benutzt, um Zyklen auszulösen oder um die Zentraleinheit anderen Tasks, welche auf dem Windows NT System laufen, zu überlassen.

ISaGRAF beenden

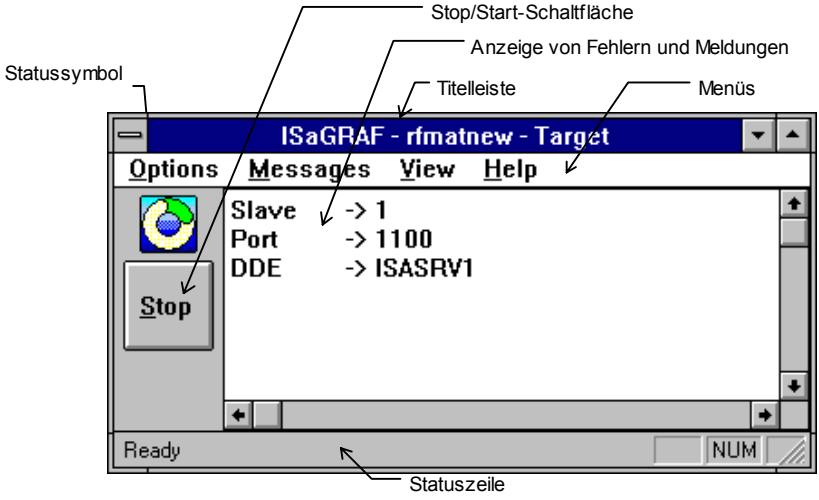
Wenn der Benutzer eine Anwendung in nicht-industriellen Umständen auf einem PC testet, kann er ISaGRAF durch Anschlagen einer Tastenkombination beenden. Der Gebrauch mehrerer Tasten sorgt für eine größere Sichereit und verhindert, daß ISaGRAF unerwartet beendet wird. Die folgende Tastenkombination wird benutzt:

Alt + F4

Eine gefährliche Nebenwirkung dieses schnellen Beendens ist, daß die E/A-Kartenschnittstelle nicht geschlossen wird. Der richtige Weg, Ihr ISaGRAF Zielsystem zu beenden, ist folgender:
- die Anwendung vom Debugger aus oder mit der Stop/Start-Schaltfläche stoppen (die E/A-Karten werden geschlossen)
- das ISaGRAF Zielsystem vom Systemmenü aus stoppen.

C.6.4. Benutzerschnittstelle

Dies ist die Benutzerschnittstelle des ISaGRAF NT Zielsystems:



Die Hauptelemente sind:
- eine Fenstertitelleiste
- eine Menüleiste
- ein Statussymbol
- eine Start/Stop-Schaltfläche
- eine Fehler- und Meldungsanzeige
- eine Statuszeile.

Der Fenstertitel enthält « ISaGRAF - name_of_appli - target », wobei name_of_appli der Name der laufenden Anwendung ist. Wenn keine Anwendung ausgeführt wird, enthält die Titelleiste « ISaGRAF - - Target ».

Menüleiste des ISaGRAF NT Zielsystems:

Die Menüleiste enthält vier Menüs:

Optionen
Meldungen
Anzeige
Hilfe

- **Menü "Optionen"**

(siehe auch Abschnitt 1 über NT:Allgemeine Informationen über Optionen)

Über das Menü "**Optionen**" wird auf die Ausführungsoptionen zugegriffen.

Folgende Optionen stehen zur Verfügung:

Slave verschafft Zugriff auf die Slave-Nummermodifizierung. Die modifizierte Option wird erst beim folgenden Start des Zielsystems aktiviert. Diese Funktion ist nicht verfügbar, wenn das Zielsystem mit mindestens einer Option in der Befehlszeile gestartet wurde.
Kommunikation verschafft Zugriff auf die Kommunikationskonfiguration. Die modifizierte Option wird erst beim folgenden Start des Zielsystems aktiviert. Diese Funktion ist nicht verfügbar, wenn das Zielsystem mit mindestens einer Option (außer der -s Option) in der Befehlszeile gestartet wurde.
DDE verschafft Zugriff auf die Modifizierung der DDE Advise Loop Rate. Die modifizierte Option wird erst beim folgenden Start des Zielsystems aktiviert. Diese Funktion ist nicht verfügbar, wenn das Zielsystem mit mindestens einer Option (außer der -s Option) in der Befehlszeile gestartet wurde.
E/A simulieren ist angekreuzt oder nicht, entsprechend dem Status der Option. Die modifizierte Option wird erst beim folgenden Stop/Start des Zielsystems aktiviert.
Priorität verschafft Zugriff auf die Modifizierung der Priorität. Die modifizierte Option wird sofort aktiviert.

Vorgabeoptionen dient der Wiedergewinnung der folgenden Vorgabeoption für die Ausführung der Anwendung:
- Kommunikation
- DDE
- Koordinaten des Bildschirmfensters

Die modifizierten Optionen werden erst beim nächsten Start des Zielsystems aktiviert. Diese Funktion ist nicht verfügbar, wenn das Zielsystem mit mindestens einer Option (außer der -s Option) gestartet wurde.

- **Menü "Meldungen"**

Das Menü "**Meldungen**" verwaltet die Ausgänge. Es enthält die folgenden Befehle:

Bestätigen stoppt das rote Blinklicht bei der Anzeige von Fehlern und Meldungen.
Löschen löscht die gesamte Ausgabe.

ISaGRAF NT Zielsystem-Symbol:

Das Symbol zeigt den Status des Zielsystems an:

- Wenn das Symbol sich dreht, befindet die Anwendung sich in der Ausführung.

- Wenn das Symbol sich nicht dreht, ist keine Anwendung vorhanden (oder die Anwendung ist gestoppt)
- Die Mitte des Symbols blinkt rot, wenn Fehler und Meldungen im Ausgabefenster angezeigt werden. Um das Blinken zu stoppen, wählen Sie « Bestätigen » im Menü « Meldungen » oder « Löschen » im selben Menü (dieser Befehl löscht die gesamten Ausgangsmeldungen). Zusätzliche Informationen zu Fehlern finden Sie im Kapitel "Fehlerverwaltung und Ausgangsmeldungen".

Die verschiedenen Zustände sind in der folgenden Tabelle zusammengefaßt:

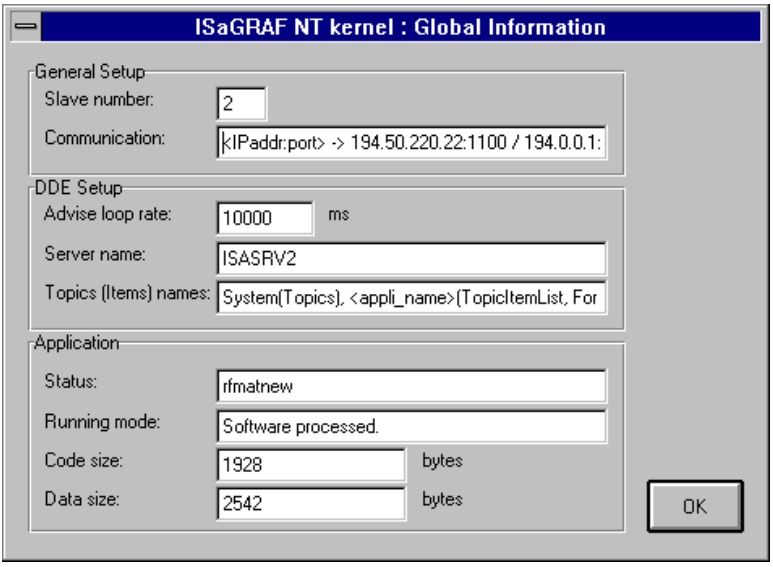
	keine Fehler	**Fehler oder Meldungen (rote Mitte)**
Anwendung in der Ausführung		
Keine Anwendung		

Start/Stop-Schaltfläche des ISaGRAF NT Zielsystems:

Die Start/Stop-Schaltfläche ist identisch mit der Start/Stop-Funktion des Debuggers. Der in der Schaltfläche angezeigte Text spiegelt den Ausführungszustand der Anwendung wieder. Wenn die Anwendung sich in der Ausführung befindet, wird « Stop » angezeigt. Wenn die Anwendung gestoppt ist (oder wenn keine Anwendung vorhanden ist), wird « Start » angezeigt. (Wenn keine Anwendung vorhanden ist und ein Startbefehl eingeleitet wird, schaltet die Schaltfläche auf den Stoppmodus und dann sofort zum Startmodus zurück).

Allgemeine Informationen über das ISaGRAF NT Zielsystem

Mit dem Befehl "Anzeige / Informationen" wird das folgende Dialogfeld angezeigt, in dem allgemeine Informationen über die Zielsystemkonfiguration und die laufende Anwendung angezeigt werden:



Die Informationen beziehen sich auf die:

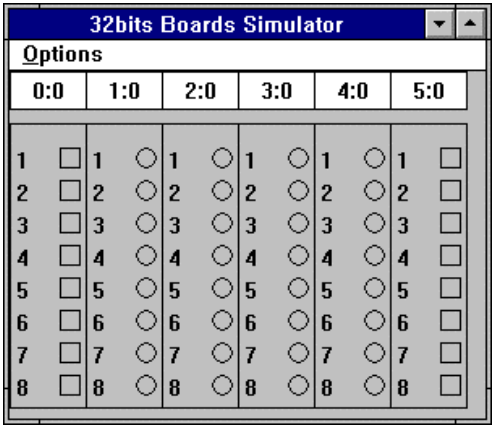
a) Allgemeine Einstellung:
   - Slave-Nummer
   - Kommunikationskonfiguration (wenn die Kommunikation Ethernet ist, wird zusätzlich zur Anschlußnummer eine Liste der verfügbaren IP-Adressen auf dem aktuellen NT System angezeigt)

b) DDE-Einstellung
   - Advise Loop Rate
   - DDE-Servername
   - DDE-Themen und Elementnamen. Dabei handelt es sich um allgemeine Informationen. Reelle Werte werden nicht wiedergespiegelt. In der Tat, die Felder zwischen < > sollten mit den echten Werten ersetzt werden.

c) Anwendung
   - Anwendungsstatus, d.h. der Name der laufenden Anwendung oder die Zeichenkette 'Keine Anwendung', wenn keine laufende Anwendung vorhanden ist.
   - Ausführungsmodus der Anwendung, der angibt, ob die Anwendung über den Software-Prozessor läuft. In diesem Fall wird « Software verarbeitet » angezeigt. Wenn die Anwendung mit einem C-Compiler kompiliert wurde, erscheint die Zeichenkette « C-kompiliert ». Wenn keine laufende Anwendung vorhanden ist, wird « Keine Anwendung » angezeigt.
   - Code-Größe in Bytes. Wenn der Ausführungsmodus « C-kompiliert » ist , ist dieses Feld null.
   - Datengröße in Bytes. Dies ist die Summe der internen Ausführungsdaten und der Variablen-Datenbank.

Simulation virtueller Karten auf dem ISaGRAF NT Zielsystem:

Wenn die Option « E/A simulieren » gewählt wird, erscheint beim nächsten Start der Anwendung das folgende Fenster:

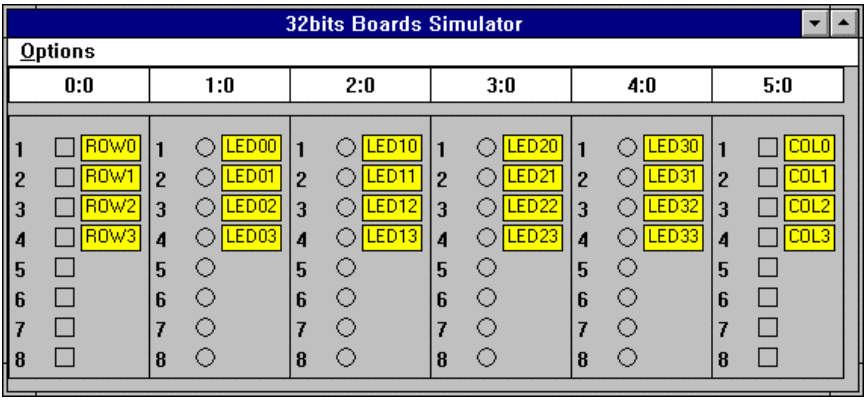


Abhängig von Ihrer E/A-Konfiguration, wird eine bestimmte Anzahl an Karten und Variablen angezeigt. Die Zahlen « s:b » oberhalb einer jeden Karten kennzeichnen die Steckplatznummer (s) und den Kartenbezeichner (b). Gezählt wird angefangen mit null (kann nicht modifiziert werden).
Das '32Bit Kartensimulator'-Fenster arbeitet mit einem Start/Stopp Anwendungsstatus. Wenn eine laufende Anwendung virtuelle Karten besitzt (oder virtuelle Karten benutzt) und wenn die Option « E/A simulieren » aktiviert ist, wird dieses Fenster angezeigt. Sobald jedoch die Schaltfläche Stopp gedrückt wird, schließt sich das Fenster. Dieses Fenster arbeitet zusammen mit den E/A-Aufrufen.

Das Menü "**Optionen**" enthält zwei Elemente:

> **Variablennamen** zeigt die Namen der Variablen an, aber nur, wenn die Symboltabelle vor dem Tic-Code geladen wird.
>
> **Hexadezimalwerte** zeigt jede Ganzzahl im Hexadezimalformat an, anstatt des vorgegebenen Dezimalformats.

Die Variablennamen sehen folgendermaßen aus:


32bits Boards Simulator
Options
0:0
1:0
2:0
3:0
4:0
5:0
ROW0
ROW1
ROW2
ROW3
LED00
LED01
LED02
LED03
LED10
LED11
LED12
LED13
LED20
LED21
LED22
LED23
LED30
LED31
LED32
LED33
COL0
COL1
COL2
COL3

C.7. "C"-Programmierung

C.7.1. Überblick

Dieses Handbuch ist Benutzern gewidmet, denen die ISaGRAF Konzepten und Workstation-Werkzeugen bereits vertraut sind. Nachdem der Benutzer reine Automatisierungsanwendungen mit Hilfe von **Umrechnungsfunktionen**, **"C"-Funktionen** und -**Funktionsbausteinen** der CJ International Standardbibliotheken erstellt hat, hat er nunmehr die Möglichkeit, "benutzerdefinierte" Umrechnungsfunktionen, "C"-Funktionen und -Funktionsbausteine zu entwickeln. Er kann die Leistungfähigkeit der ISaGRAF Zielsteuerung steigern, indem er neue Bibliotheken erstellt, und die Workstation und Hardware noch flexibler gestalten.

Mit einem "C"-Entwicklungssystem, etwas Erfahrung auf dem Gebiet der "C"-Programmierung und unter Anwendung dieses Handbuchs kann der Benutzer seine ISaGRAF Zielsteuerung seinen Ansprüchen entsprechend persönlich gestalten. Solche Entwicklungen steigern nicht nur die Leistungsfähigkeit einer Zielsteuerung, sondern verbessern auch den Komfort und die Qualität der ISaGRAF Workstation-Entwicklungen für den Automatisierungstechniker.

Die in diesem Handbuch enthaltenen Informationen sind keinem bestimmten Zielsystem gewidmet. Manche Funktionen (wie beipielsweise Multitasking-Kapazitäten) können allerdings auf bestimmten Monotasking-Systemen nicht implementiert werden.

Standardfunktionen der ISaGRAF Workstation

Die ISaGRAF Workstation verfügt über zahlreiche Funktionen für die Verwaltung der " C "-Bibliotheken in der Automatisierungsentwicklung, in der " C " - Umrechnungsfunktionen, -Funktionen oder - Funktionsbausteine lediglich durch ihre Schnittstellen definierte "**unbekannte Größen**" sind.

Der ISaGRAF Bibliotheksmanager wird benutzt, um bestehenden Bibliotheken neue Elemente hinzuzufügen und um die Schnittstelle zwischen der "C"-Implementierung und der Benutzung dieser Elemente in der **ST**/**FBS**-Programmierung zu definieren. Außerdem verfügt er über Funktionen für die automatische Entwicklung von "C"-Quellcodeblocks für Umrechnungen, Funktionen und Funktionsbausteine und besitzt Werkzeuge für die Bearbeitung solcher "C"-Quelldateien. Siehe **ISaGRAF Benutzerhandbuch** für zusätzliche Informationen über die Funktionen des Bibliotheksmanagers.

Entwicklungen in der "C"-Sprache

Die ISaGRAF Workstation verfügt weder über einen "C"-Compiler, noch über ein Querentwicklungswerkzeug. Der Benutzer braucht deshalb einen dem ISaGRAF Zielsystem dedizierten "C"-Compiler, um seine "C"-Komponenten in den ISaGRAF Kernel integrieren zu können.

Wenn ein Cross-Compiler benutzt wird, hat der Benutzer in der ISaGRAF Workstation die Möglichkeit, eine benutzerdefinierte MS-DOS Befehlszeile (.bat) in einem DOS-Fenster auszuführen. Der Cross-Compiler muß in einem DOS-Emulationsfenster laufen, sonst muß

Windows geschlossen werden, um die Compiler und Linker in einem reinen MS-DOS Kontext auszuführen.

Datenblätter

Mit dem ISaGRAF Bibliotheksmanager kann der Benutzer eine Textbeschreibung für jedes Bibliothekselement erfassen. Dieses sogenannte **Datenblatt**, eine Gebrauchsanweisung des entwickelten "C"-Komponenten, ist für den Automatisierungstechniker bestimmt und beschreibt, wie die entsprechende Umrechnungen, Funktion oder der Funktionbaustein in einer ISaGRAF Anwendung benutzt wird.

Es ist wichtig, daß die betreffende Umrechnung, "C"-Funktion oder der "C"-Funktionsbaustein in allen Einzelheiten beschrieben wird, so daß das Element auch wirklich vom Automatisierungstechniker als eine gebrauchsfertige ISaGRAF Funktion benutzt werden kann. Das Datenblatt einer "C"-Funktion sollte folgendes beeinhalten:

- ❑ die genaue Beschreibung der ausgeführten Funktion
- ❑ die ausführliche Beschreibung der Aufrufparameter
- ❑ die Bedeutung des Returnparameters
- ❑ die genaue Typbeschreibung der Aufruf- und Returnparameter
- ❑ den Anwendungskontext

Das Datenblatt eines "C"-Funktionsbausteins sollte folgendes beeinhalten:

- ❑ die genaue Beschreibung der bei der Aktivierung des Bausteins ausgeführten Funktion
- ❑ die ausführliche Beschreibung der Aufrufparameter
- ❑ die Bedeutung der Returnparameter
- ❑ die genaue Typbeschreibung der Aufruf- und Returnparameter
- ❑ den Anwendungskontext

Das Datenblatt einer Umrechnungsfunktion sollte folgendes beeinhalten:

- ❑ die genaue Bedeutung der Umrechnung, wenn sie auf eine Eingangsvariable angewandt wird
- ❑ die genaue Bedeutung der Umrechnung, wenn sie auf eine Ausgangsvariable angewandt wird
- ❑ die Grenzwerte und den Anwendungskontext

Datenblätter können ferner die folgenden Informationen enthalten:

- ❑ die genaue Kennzeichnung des Elements
- ❑ Informationen über seine Entwicklung und Instandhaltung
- ❑ das unterstützte Zielsystem
- ❑ spezielle Multitasking-Funktionalitäten
- ❑ Anforderungen hinsichtlich Systemservice, Speicher, Treiber...

C.7.2. "C"-Umrechnungsfunktionen

Die ISaGRAF Workstation beinhaltet eine **lineare Umrechnungs-Utility**, die während der Programmausführung einfache analoge E/A-Umrechnungen auf der ISaGRAF Zielsteuerung tätigt. Diese Utility benötigt keine "C"-Entwicklung und beschränkt sich auf streng steigende

oder fallende kontinuierliche Funktionen. Siehe ISaGRAF Benutzerhandbuch für eine ausführliche Beschreibung dieser Werkzeuge.

Die Umrechnungsfunktionen ermöglichen dem Benutzer, eine beliebige komplexe Umrechnung, die spezifische in der "C"-Sprache geschriebene Operationen einschließt, anzuwenden. Grundsätzlich werden Umrechnungsfunktionen sowohl für die **Eingänge** als auch für die **Ausgänge** definiert. Selbst wenn eine der Richtungen nicht benutzt wird, sollte sie dennoch implementiert und getestet werden, bevor die Umrechnung in den ISaGRAF Kernel integriert wird, um Systemabstürze aufgrund eines falschen Aufrufs zu vermeiden.

Die Umrechnungsfunktionen werden in der "C"-Sprache geschrieben, kompiliert und mit dem ISaGRAF Kernel verbunden. Der neue Kernel muß auf der ISaGRAF Zielsteuerung installiert werden, bevor die neuen Umrechnungsfunktionen in den ISaGRAF Projekten benutzt werden können. Da diese Umrechnungsfunktionen nicht in den ISaGRAF Simulator integriert werden können, müssen die ISaGRAF Anwendungen simuliert werden, **bevor** die nicht-standard Umrechnungsfunktionen hinzugefügt werden.

Der "C"-Quellcode der von CJ International geschriebenen Standardumrechnungen wird zusammen mit der ISaGRAF Workstation installiert. Diese Umrechnungen können als Beispiele für die Erstellung neuer Funktionen benutzt werden, sollten jedoch **nicht modifiziert** werden, damit sie auch weiterhin in zukünftigen ISaGRAF Anwendungen benutzt werden können. Die mit der ISaGRAF Workstation installierten Standardumrechnungen werden vom ISaGRAF Simulator unterstützt.

Warnung: Umrechnungsfunktionen sind **synchrone** Operationen, die bei der Ausführung vom ISaGRAF E/A-Manager während der Ein- und Ausgangsphasen des Anwendungszylus aktiviert werden. Die für die Ausführung einer Umrechnungsfunktion benötigte Zeit ist in der **Zykluszeit** der ISaGRAF Anwendung enthalten. Der Benutzer sollte sich vergewissern, daß die Umrechnungsfunktion keine "Warteoperationen" enthält, so daß der ISaGRAF Zyklus nicht unnötigt ausgedehnt wird.

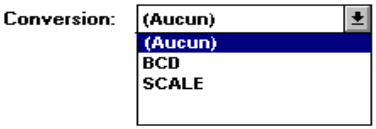
Hinzufügen einer Funktion in die ISaGRAF Bibliothek

Um der ISaGRAF Bibliothek eine neue Umrechnungsfunktion der Workstation hinzuzufügen, benutzt man den ISaGRAF Bibliotheksmanager. Öffnen Sie die Bibliothek der Umrechnungsfunktionen und leiten Sie den Befehl "**Neu**" im "**Datei**"-Menü ein. Es ist nicht nötig, die Parameter in der Workstation zu definieren, da Umrechnungsfunktionen eine vordefinierte Standardschnittstelle benutzen.

Nach der Erstellung einer neuen Umrechnungsfunktion muß ihr **Datenblatt** abgefaßt werden. Der Block des "C"-Quellcodes für die neue Umrechnungsfunktion wird automatisch vom ISaGRAF Bibliotheksmanager generiert.

Gebrauch einer Umrechnung in einem ISaGRAF Projekt

Die definierten Umrechnungsfunktionen können zum Filtern beliebiger Ein- und Ausgangsvariablen des ausgewählten Projekts benutzt werden. Um einer Variablen eine Umrechnungsfunktion zuzuordnen, wird der Variablendeklarationseditor gestartet. Wählen Sie eine analoge Ein- oder Ausgangsvariable und bearbeiten Sie deren Parameter. Das Feld "**Umrechnung**" im Dialogfeld für die Deklaration analoger Variablen dient der Auswahl einer Umrechnungsfunktion, die der analogen E/A-Variablen zugeordnet wird:



In der Liste erscheinen sowohl Umrechnungsfunktionen als auch Umrechnungstabellen. Folglich kann einer Tabelle nicht der gleiche Name gegeben werden, wie einer Funktion. Variablen können nur solchen Umrechnungsfunktionen zugeordnet werden, welche bereits definiert oder in den ISaGRAF Kernel integriert worden sind.

Standard-"C"-Schnittstelle

Die Schnittstelle einer Umrechnungsfunktion hat stets das gleiche Format. Die Aufruf- und Returnparameter werden in einer Struktur erfaßt. Diese Struktur wird in der Datei "**TACN0DEF.h**" definiert:

```
/*
  Name: tacn0def.h
  Definitionsdatei für Umrechnungsfunktionen
*/
#define DIR_INPUT 0              /* Richtung = Eingangsumrechnung  */
#define DIR_OUTPUT 1             /* Richtung = Ausgangsumrechnung */

typedef int32   T_ANA;           /* ganzzahliger ANA Typ           */
typedef float   T_REAL;          /* realer ANA Typ                 */
typedef struct {                 /* Umrechnungsstruktur            */
    uint16 number;               /* Umrechnungsnummer (reserviert)  */
    uint16 direction;            /* Umrechnungsrichtung            */
    T_REAL *before;              /* Wert vor der Umrechnung        */
    T_REAL *after;               /* Wert nach der Umrechnung       */
  } str_cnv;

#define ARG_BEFORE (*(arg->before))
#define ARG_AFTER  (*(arg->after))
#define DIRECTION  (arg->direction)
/* eof */
```

Die "**str_cnv**" Struktur ist eine abgeschlossene Beschreibung der Schnittstelle. Der einzige Parameter einer "C"-Umrechnungsfunktion ist ein Pointer zu einer solchen Struktur. Das Feld "**number**" ist die logische Nummer der Umrechnungsfunktion (Lokalisierung in der ISaGRAF Bibliothek) und braucht nicht in der Programmierung benutzt zu werden.
Das Feld "**direction**" gibt an, ob die Umrechnung auf eine Eingangs- oder Ausgangsvariable angewendet werden soll. Bei einer Eingangsumrechnung enthält es den Wert **DIR_INPUT**, bei einer Ausgangsumrechnung den Wert **DIR_OUTPUT**.

Das Feld "**before**" zeigt auf den Wert vor der Umrechnung (Pointer). Dieses Feld hat bei einer Eingangsumrechnung eine unterschiedliche Bedeutung wie bei einer Ausgangsumrechnung. Bei einer Eingangsumrechnung stellt es den elektrischen (auf dem Sensor gelesenen) Wert dar, wenn das Feld **direction** den Wert **DIR_INPUT** hat. Bei einer Ausgangsumrechnung stellt es den physischen (in den programmierten Gleichungen benutzen) Wert dar, wenn das Feld **direction** den Wert **DIR_OUTPUT** hat.

Das Feld "**after**" zeigt auf den Wert nach der Umrechnung (Pointer). Dieses Feld hat bei einer Eingangsumrechnung eine unterschiedliche Bedeutung wie bei einer Ausgangsumrechnung. Bei einer Eingangsumrechnung stellt es den physischen (in den programmierten Gleichungen benutzen) Wert dar, wenn das Feld **direction** den Wert **DIR_INPUT** hat. Bei einer Ausgangsumrechnung stellt es den elektrischen (auf dem Sensor gelesenen) Wert dar, wenn das Feld **direction** den Wert **DIR_OUTPUT** hat.

Der Programmierer kann die Definitionen "**ARG_BEFORE**" und "**ARG_AFTER**" benutzen, um direkten Zugriff auf die Felder **before** und **after** der an die "C"-Umrechnungsfunktion gegebenen Struktur zu erhalten. Die verarbeiteten Werte sind **Gleitpunktwerte mit einfacher Präzision**. Das Ergebnis wird zu einer langen Ganzzahl umgerechnet, wenn die Umrechnung auf eine analoge ganzzahlige Variable angewendet wird. Folglich kann die gleiche Umrechnung sowohl für reale und als auch für ganzzahlige analoge E/A-Variablen benutzt werden.

Quellcode

Da die Umrechnungsfunktion sowohl für analoge Eingangsvariablen, als auch für analoge Ausgangsvariablen benutzt werden kann, wird der "C"-Quellcode der Funktion in zwei Hauptabschnitte aufgeteilt: die Eingangsumrechnung und die Ausgangsumrechung. Das Feld **direction** der Schnittstellenstruktur wird benutzt, um die anzuwendende Umrechnung auszuwählen. Wenn eine neue Umrechnungsfunktion erstellt wird, generiert der ISaGRAF Bibliotheksmanager den kompletten Codeblock der Funktion automatisch, einschließlich der "**IF**"-Struktur. Es folgt der Standardblock einer Umrechnungsfunktion:

```
/*
  Umrechnungsfunktion
  Name: sample
*/
#include <tasy0def.h>
#include <tacn0def.h>

void CNV_sample (str_cnv *arg)
{
        if (DIRECTION == DIR_INPUT) { /*EINGANGSUMRECHNUNG*/
        }
        else { /*AUSGANGSUMRECHNUNG*/
        }
}
```

/* Die folgende Funktion veranschaulicht die Verbindung mit dem ISaGRAF E/A-Manager, unter Anwendung des Umrechnungsnamens. Diese Funktion wird vollständig vom ISaGRAF Bibliotheksmanager generiert. */

```
UFP cnvdef_sample (char *name)
{
        sys_strcpy (name, "SAMPLE");       /*liefert den Namen der Umrechnung
*/
        return (CNV_sample);        /* gibt die Implementierungsfunktion zurück */
}
```

Um den automatisch generierten Block zu vervollständigen, sollten für die Ein- und Ausgangsumrechnungen zwei separate lokale Funktionen geschrieben werden. Wie im obigen Beispiel veranschaulicht, können diese Funktionen vom Haupt-Algorithmus in der **IF**-Struktur aufgerufen werden.

Die "**TASY0DEF.H**" Include-Datei des ISaGRAF Kernels wird für die system-abhängigen Definitionen gebraucht. Ferner enthält sie die Definition des **UFP**-Typs, der ein Pointer zu einer Void-Funktion darstellt und für die Deklaration der Funktion benutzt wird.

Verbindungen zwischen Projekten und "C"-Implementierung

Die logische Verbindung zwischen der Implementierung einer Umrechnungsfunktion und der Benutzung der Umrechnung in einem ISaGRAF Projekt erfolgt mit dem Namen der Umrechnung. Eine "Deklarationsfunktion" wird dem "C"-Quellcode der Umrechnungsfunktion beigefügt. Diese Funktion wird nur ein einziges Mal beim Start der Anwendung aufgerufen und teilt dem ISaGRAF E/A-Manager den Namen der Umrechnung mit, welcher der zu implementierenden Funktion entspricht. Hier das Standardformat einer solchen Deklarationsfunktion:

```
UFP cnvdef_xxx (char *name)
{
        strcpy (name, "XXX");       /* liefert den Namen der Umrechnung */
        return (CNV_xxx);           /* gibt die Implementierungsfunktion zurück */
}
/* (xxx ist der Name der Umrechnung) */
```

In der **strcpy** Anweisung wird der Funktionsname **großgeschrieben**. In der Implementierungsfunktion und in der Deklarationsfunktion muß er kleingeschrieben werden.

Durch die Benutzung der Vorzeichen "**CNV_**" und "**cnvdef_**" für die Implementierungs- und Definitionsfunktionen können Umrechnungen erstellt werden, die den gleichen Namen wie ein Schlüsselwort der "C"-Sprache oder wie eine bereits in den ISaGRAF "C"-Bibliotheken definierte Funktion tragen.

Der Deklarationsfunktion können andere Anweisungen hinzugefügt werden, um eine beliebige, spezifische, diese Umrechnung betreffende Initialisierungsoperation auszuführen.

Das ISaGRAF System bewirkt, daß diese Funktion **nur einmal** beim Anwendungsstart aufgerufen wird.

Die Deklarationsfunktion wird für alle integrierten Umrechnungsfunktionen aufgerufen, selbst wenn sie nicht in der ISaGRAF Anwendung benutzt werden. Der ISaGRAF Kernel generiert einen fatalen Fehler, wenn eine nicht in den Kernel integrierte Umrechnung in einer Anwendung benutzt wird.

Bevor neue Funktionen mit dem Kernel verbunden werden können, muß der Benutzer eine neue "C"-Quelldatei namens "**GRCN0LIB.C**" schreiben und in die Dateiliste für den Linker einfügen (mit den festgehaltenen Umrechnungsfunktionen). "**GRCN0LIB.C**" enthält lediglich ein Feld mit Deklarationsfunktionen. Dieses Feld wird bei Anwendungsinitialisierungen gelesen, um eine dynamische Verbindung mit denen in "C" geschriebenen Umrechnungsfunktionen herzustellen. Hier ein Beispiel:

```
/* Datei "GRCN0LIB.c" - Beispiel mit Umrechnungen der Standardbibliothek */

#include <tasy0def.h>                /* erforderlich für die Typdefinition */

extern UFP cnvdef_scale (char *name);    /* Deklarationsfunktion für SCALE
Umrechnung */

extern UFP cnvdef_bcd (char *name);      /* Deklarationsfunktion für BCD
Umrechnung */

UFP_LIST CNVDEF[ ] = {          /* Feld von Deklarationsfunktionen für */
        /* integrierte Umrechnungsfunktionen */
        cnvdef_scale,
        cnvdef_bcd,

NULL };

/* Dateiende */
```

Das Feld **CNVDEF** muß in einem NULL Pointer enden, sonst könnten schwerwiegende Probleme auftreten. Wenn das Feld nicht definiert wird, treten ungelöste Referenzen auf, wenn der neue ISaGRAF Kernel verbunden wird.

Diese Datei ermöglicht den Aufbau eines neuen Kernels, einschließlich aller bestehenden Umrechnungen. Es ist ebenfalls möglich, einen projekt-dedizierten Kernel zu erstellen, indem nur Umrechnungen eines bestimmten Projekts in das **CNVDEF** Feld aufgenommen werden. Die Datei "**GRCN0LIB.C**" wird automatisch bei der Entwicklung des Anwendungscodes vom ISaGRAF Codegenerator generiert. Die Datei wird im ISaGRAF Projektverzeichnis gespeichert und enthält lediglich die Umrechnungen des Projekts.

Begrenzungen

Die ISaGRAF Bibliothek kann bis zu **128** Umrechnungsfunktionen enthalten. In einer Umrechnungsfunktion können beliebige Operationen verarbeitet werden. Man sollte jedoch nicht vergessen, daß die Funktionen im ISaGRAF Zyklus **synchron** aufgerufen werden, so daß die Ausführung der Funktion einen direkten Einfluß auf die Zykluszeit nimmt.

C.7.3. "C"-Funktionen

"C"-Funktionen werden benutzt, um die Standardkapazitäten der **ST**- und **FBS**-Sprachen zu erweitern. Sie können für spezifische Berechnungen, Systemaufrufe, Kommunikationen und die Installation einer Servicegruppe für den Dialog zwischen einer ISaGRAF Anwendung und anderen Tasks benutzt werden. Funktionen werden in der "C"-Sprache geschrieben, kompiliert und mit dem ISaGRAF Kernel verbunden. Der verbesserte Kernel muß auf der ISaGRAF Zielsteuerung installiert werden, bevor die neuen Funktionen in den ISaGRAF Projekten benutzt werden können.

Da die neuen Funktionen nicht in den ISaGRAF Simulator integriert werden können, müssen die ISaGRAF Anwendungen simuliert werden, **bevor** nicht-standard Funktionen benutzt werden.

Warnung: Funktionen stellen **synchrone** Operationen dar, die vom ISaGRAF Kernel während der Ausführung eines Anwendungszyklus aktiviert werden. Die für die Ausführung einer Funktion verwendete Zeit ist in der **Zykluszeit** der ISaGRAF Anwendung inbegriffen. Der Benutzer sollte darauf achten, daß keine "Warteoperationen" in einer Funktion programmiert sind, so daß die ISaGRAF Zyklusverarbeitung nicht unnötig ausgedehnt wird.

Hinzufügen einer Funktion in die ISaGRAF Bibliothek

Um der ISaGRAF Bibliothek eine neue Funktion der Workstation hinzuzufügen, benutzt man den ISaGRAF Bibliotheksmanager. Öffnen Sie die Bibliothek der Funktionen und leiten Sie den Befehl "**Neu**" im "**Datei**"-Menü ein. Nach der Erstellung einer neuen Funktion muß ihr **Datenblatt** abgefaßt werden. Der Block des "C"-Quellcodes für die neue Funktion wird automatisch vom ISaGRAF Bibliotheksmanager generiert.

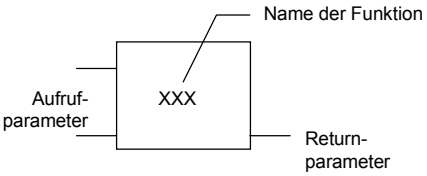
Gebrauch einer "C"-Funktion in einem ISaGRAF Projekt

Beliebige integrierte "C"-Funktionen können als Standardfunktionen in den Programmen eines ISaGRAF Projekts benutzt werden. "C"-Funktionen können von den **ST**- und **FBS**-Sprachen und von speziellen Anweisungen der **AS**-Sprache aufgerufen werden.

Der Aufruf einer "C"-Funktion von der **ST**-Sprache folgt den Funktionsaufrufkonventionen dieser Sprache. Die Aufrufparameter der Funktion werden hinter den Namen der Funktion in Klammern geschrieben und durch Kommas getrennt. Der Ausdruck stellt den von der Funktion zurückgegebenen Wert dar. "C"-Funktionsaufrufe können in beliebige Zuweisungsanweisungen und komplexe Ausdrücke eingefügt werden. Hier das Beispiel eines "C"-Funktionsaufrufs in einer Zuweisungsanweisung:

Ergebnis := FunkName (par1, par2, ... parN);

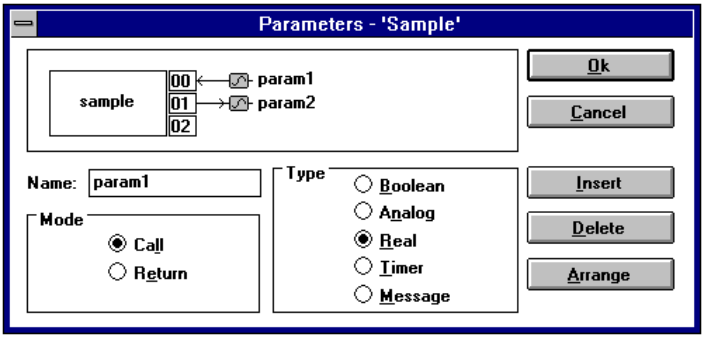
FBS-Programme können beliebige "C"-Funktionen aufrufen. Eine Funktion wird wie eine Standardfunktionsbox benutzt. Ihre Aufrufparameter werden links mit der Funktionsbox verbunden, der Returnparameter rechts. Hier das Standardformat einer solchen Funktionsbox:



Eine "C"-Funktion kann von einem beliebigen AS-Aktionsblock oder in einer beliebigen, mit einer Transition verbundenen, booleschen Bedingung aufgerufen werden.

Schnittstellendefinition einer "C"-Funktion

Der Befehl "**Parameter**" im "**Datei**"-Menü öffnet ein Dialogfeld, in dem die Aufruf- und Returnparameter einer neuen "C"-Funktion definiert werden. Eine Funktion kann bis zu **31** Aufrufparameter, jedoch nur **einen einzigen** Returnparameter besitzen.



Im oberen Teil des Fensters erscheinen die Parameter der "C"-Funktion in der Reihenfolge des Funktionsaufruf-Prototyps: zuerst die Aufrufparameter und zuletzt der Returnparameter. Im unteren Teil des Fensters erscheint eine detaillierte Beschreibung des derzeit in der Liste ausgewählten Parameters:

- Name des Parameters
- Richtung (Aufruf/Return) des Parameters
- Typ des Paramters

Sämtliche ISaGRAF Datentypen können benutzt werden: boolesch, ganzzahlig analog, real analog, Timer oder Zeichenkette. Zwischen ganzzahlig analog und real analog wird unterschieden.

Es folgt die Entsprechung zwischen ISaGRAF Typen und "C"-Typen:

BOOLESCH	unsigned long	vorzeichenloses 32 Bit-Wort: 1=true / 0=false
ANALOG	long	vorzeichenbehaftetes ganzzahliges 32 Bit-Wort

REAL	float	Gleitpunktwert mit einfacher Präzision
TIMER	unsigned long	vorzeichenloses ganzzahliges 32 Bit-Wort (Einheit = 1 Millisekunde)
ZEICHENKETTE	char *	Zeichenkette

Wenn ein Zeichenkettenwert an eine "C"-Funktion gegeben wird, kann er keine Nullzeichen enthalten, denn die an den "C"-Code gegebene Zeichenkette endet in null. Vergessen Sie nicht, daß der Returnparameter an letzter Stelle in der Liste erscheinen muß. Bei der Parameterbenennung müssen die folgenden Regeln beachtet werden:

- Der Name darf nicht länger als 16 Zeichen sein.
- Das erste Zeichen muß ein Buchstabe sein.
- Die weiteren Zeichen müssen Buchstaben, Zahlen oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Ein Parametername darf nicht von mehreren Parametern einer Funktion benutzt werden. Ein Aufrufparameter darf nicht den gleichen Namen wie ein Returnparameter haben. Gleiche Namen können hingegen für Parameter **unterschiedlicher** Funktionen benutzt werden. Der Vorgabename für den Returnparameter ist "**Q**". Dieser Name kann frei modifiziert werden. Parameternamen werden benutzt, um die betreffenden Parameter im "C"-Quellcode zu identifizieren.

Mit dem Befehl "**Einfügen**" kann ein neuer Parameter vor dem ausgewählten Parameter eingefügt werden. Der Befehl "**Löschen**" löscht den ausgewählten Parameter. Der Befehl "**Arrangieren**" sortiert die Parameter automatisch und plaziert den Returnparameter an das Ende der Liste. Klicken Sie auf die Schaltfläche "**OK**", um die Definitionen der Funktionsschnittstelle zu speichern und das Dialogfeld zu schließen. Klicken Sie auf "**Abbrechen**", um das Dialogfeld zu schließen, ohne die Modifikationen zu speichern.

"C"-Schnittstelle einer Funktion

Die Schnittstelle einer Funktion ist abhängig von der Definition ihrer Parameter. Aufruf- und Returnparameter werden in eine Struktur gegeben. Diese Struktur wird in der Datei "**GRUS0nnn.H**" definiert, in der "**nnn**" die logische Nummer der Funktion in der ISaGRAF Bibliothek ist. Hier ein Beispiel einer "C"-Schnittstelle für die "**SIN**" Funktion (Sinusberechnung):

```
/* Datei: GRUS0255.h - Funktion "sample" */
typedef  long                   T_BOO;
typedef  long                   T_ANA;
typedef  float                  T_REAL;
typedef  long                   T_TMR;
typedef  char                   *T_MSG;

typedef struct {
        /* AUFRUF   */          T_REAL   _param1;
        /* RETURN */            T_REAL   _param2;
```

```
} str_arg;

#define PARAM1          (arg->_param1)
#define PARAM2          (arg->_param2)

/* Dateiende */
```

Es folgen die Entsprechungen zwischen den ISaGRAF Typen und den "C"-Typen. Die ISaGRAF Typen werden in der Definitionsdatei der Funktion als "C"-Typen definiert.

boolesch	T_BOO	long (32 Bits)
ganzzahlig analog	T_ANA	long
real analog	T_REAL	float (32 Bits - einfache Präzision)
Timer	T_TMR	long
Zeichenkette	T_MSG	char * (32 Bits - Zeichenpointer)

Jedes Feld der "**str_arg**" Struktur entspricht einem Parameter der Funktion. Der Returnparameter ist der letzte in der Struktur. Die Aufrufparameter erscheinen in der Struktur in der gleichen Reihenfolge wie in der Funktionsdefinition. Für jeden Parameter wird ein Bezeichner in Großbuchstaben definiert, der den direkten Zugriff auf einen an die "C"-Funktion gegebenen Parameter der Struktur herstellt. Die Namen der Bezeichner sind jene, welche in der Definition der Funktion mit dem ISaGRAF Bibliotheksmanager eingegeben wurden.

Die "C"-Definitionsdatei wird jedesmal aktualisiert, wenn die Funktionsschnittstelle im ISaGRAF Bibliotheksmanager modifiziert wird. Dies sichert die Konsistenz zwischen der Implementierung der Funktion und ihrer Verwendung in den Programmen der ISaGRAF Anwendungen.

Quellcode

Es folgt der Standardblock einer "C"-Funktionsimplementierung:

```
/* Beispiel einer Benutzerfunktion - Nummer: "255" - Name: "SAMPLE" */

#include "tasy0def.h"           /* allgemeine ISaGRAF Kernel-Definitionen */
#include "grus0255.h"           /* Schnittstellendefinition der Funktion 255 */

void USP_sample (str_arg *arg)
{
        /* Rumpf der Funktion */
}

/* Die folgende Funktion wird für die Initialisierung der Funktion und die
Deklaration ihrer Implementierung benutzt. Sie stellt die Verbindung mit dem
```

ISaGRAF Kernel unter Anwendung des Funktionsnamens her. Diese Funktion wird automatisch vom ISaGRAF Bibliotheksmanager generiert. */

```
UFP uspdef_sample (char *name)
{
        strcpy (name, "SAMPLE");/* liefert den Namen der Funktion */
        return (USP_sample);        /* gibt die Implementierungsfunktion zurück */
}

/* Dateiende */
```

Die "**TASY0DEF.H**" Include-Datei des ISaGRAF Kernels wird für die system-abhängigen Definitionen gebraucht. Ferner enthält sie die Definition des **UFP**-Typs, der ein Pointer zu einer Void-Funktion darstellt und für die Deklaration der Funktion benutzt wird.

Verbindungen zwischen Projekten und "C"-Implementierung

Die logische Verbindung zwischen der Implementierung einer "C"-Funktion und ihrer Benutzung in den Programmen eines ISaGRAF Projekts erfolgt mit dem Namen der Funktion. Eine "Deklarationsfunktion" wird dem "C"-Quellcode der Funktion beigefügt. Diese Funktion wird nur ein einziges Mal beim Start der Anwendung aufgerufen und teilt dem ISaGRAF Kernel den Namen der "C"-Funktion mit, welche der implementierten Funktion entspricht. Hier das Standardformat einer solchen Deklarationsfunktion:

```
UFP uspdef_xxx (char *name)
{
        strcpy (name, "XXX");/* liefert den Namen der Funktion */
        return (USP_xxx);           /* gibt die Implementierungsfunktion zurück */
}
/* (xxx ist der Name der Funktion) */
```

In der **strcpy** Anweisung muß der Name der Funktion **großgeschrieben** werden. In der Implementierungsfunktion und in der Deklarationsfunktion muß er kleingeschrieben werden. Durch die Benutzung der Vorzeichen "**USP_**" und "**uspdef_**" für die Implementierungs- und Definitionsfunktionen können Funktionen erstellt werden, die den gleichen Namen wie ein Schlüsselwort der "C"-Sprache oder wie eine bereits in den ISaGRAF "C"-Bibliotheken definierte Funktion tragen.

Der Deklarationsfunktion können andere Anweisungen hinzugefügt werden, um eine beliebige, spezifische, diese Funktion betreffende Initialisierungsoperation auszuführen. Das ISaGRAF System bewirkt, daß diese Funktion **nur einmal** beim Anwendungsstart aufgerufen wird. Die Deklarationsfunktion wird für alle integrierten Funktionen aufgerufen, selbst wenn sie nicht in der ISaGRAF Anwendung benutzt werden. Der ISaGRAF Kernel generiert einen fatalen Fehler, wenn eine nicht in den Kernel integrierte "C"-Funktion in einer Anwendung benutzt wird.

Bevor neue Funktionen mit dem Kernel verbunden werden können, muß der Benutzer eine neue "C"-Quelldatei namens "**GRUS0LIB.C**" schreiben und in die Dateiliste für die Verbindung

einfügen (mit den festgehaltenen Funktionen). "**GRUS0LIB.C**" enthält lediglich ein Feld mit Deklarationsfunktionen. Dieses Feld wird bei der Anwendungsinitialisierung gelesen, um eine dynamische Verbindung mit den in "C" geschriebenen Funktionen herzustellen. Hier ein Beispiel:

```
/* Datei "GRUS0LIB.c" - Beispiel mit trigonometrischen Funktionen */

#include <tasy0def.h>                /* erforderlich für die Typendefinition */

extern UFP uspdef_fc1 (char *name);        /* Deklarationsfunktionen */
extern UFP uspdef_fc2 (char *name);
extern UFP uspdef_fc3 (char *name);
extern UFP uspdef_fc4 (char *name);

UFP_LIST USPDEF[ ] = {            /* Feld der Deklarationsfunktionen  */
        /* für integrierte Funktionen */
        uspdef_fc1,
        uspdef_fc2,
        uspdef_fc3,
        uspdef_fc4,

NULL };

/* Dateiende */
```

Das Feld **USPDEF** muß in einem NULL Pointer enden, sonst könnten schwerwiegende Probleme auftreten. Wenn das Feld nicht definiert wird, treten ungelöste Referenzen auf, wenn der neue ISaGRAF Kernel verbunden wird. Diese Datei ermöglicht den Aufbau eines neuen Kernels, einschließlich aller bestehenden Funktionen. Es ist ebenfalls möglich, einen projekt-dedizierten Kernel zu erstellen, indem nur Funktionen eines bestimmten Projekts in das **USPDEF** Feld aufgenommen werden. Die Datei "**GRUS0LIB.C**" wird automatisch bei der Entwicklung des Anwendungscodes vom ISaGRAF Codegenerator generiert. Die Datei wird im ISaGRAF Projektverzeichnis gespeichert und enthält lediglich die Funktionen des Projekts.

Begrenzungen

Die ISaGRAF Bibliothek kann bis zu **255** "C"-Funktionen enthalten. In einer Funktion können beliebige Operationen verarbeitet werden. Man sollte jedoch nicht vergessen, daß die Funktionen im ISaGRAF Zyklus **synchron** aufgerufen werden, so daß ihre Ausführung einen direkten Einfluß auf die Zykluszeit nimmt.

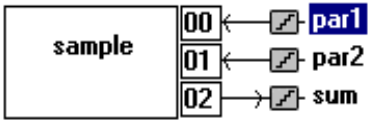
Vollständiges Beispiel

Es folgt das vollständige Beispiel einer "**sample**" Funktion, die eine Addition ausführt. Hier das Datenblatt der Funktion:

Name:	SAMPLE

Beschreibung:	führt eine ganzzahlige analoge Addition aus
Erstellungsdatum:	1.7.1992
Autor:	CJ International
Aufruf:	par1, par2: ganzzahlige Operanden
Return:	ganzzahlige Summe
Prototyp:	sum := sample (par1, par2);

Es folgt die Schnittstelle der Funktion:



Es folgt der "C"-Quellcode-Vorsatz der Funktion:

```
/* Datei: GRUS0255.h - benutzerdefinierte "C"-Funktion - Name: sample */

/* Definition der ISaGRAF Standarddatentypen */

typedef long  T_BOO;
typedef long  T_ANA;
typedef float T_REAL;
typedef long  T_TMR;
typedef char  *T_MSG;

/* Definition der Struktur der Aufruf- und Returnparameter */

typedef struct {
        T_ANA _par1;            /* Aufrufparameter #1 */
        T_ANA _par2;            /* Aufrufparameter #2 */
        T_ANA _sum;             /* Returnparameter */
} str_arg;

/* Bezeichner für den Zugriff auf die Aufruf- und Returnparameter */

#define PAR1                    (arg->_par1)
#define PAR2                    (arg->_par2)
#define SUM                     (arg->_sum)

/* Dateiende */
```

Es folgt der "C"-Quellcode der Funktion. Nur die fettgedruckten Zeilen wurden manuell vom C-Programmierer eingegeben.

```
/* Datei: GRUS0255.c - benutzerdefinierte C-Funktion - Name: SAMPLE */

#include "tasy0def.h"          /* erforderlich für die Typendefinition */
#include "grus0255.h"          /* C-Quellcode-Vorsatz der Funktion*/

/* C-Hauptservice: berechnet die Addition */

void USP_sample (str_arg *arg)
{
        SUM = PAR1 + PAR2;
}

/* Deklarationsservice - erforderlich für die dynamische Verbindung mit dem
ISaGRAF Kernel */

UFP uspdef_sample (char *name)
{
        strcpy (name, "SAMPLE");
        return (USP_sample);
}
/* Dateiende */
```

C.7.4. "C"-FUNKTIONSBAUSTEINE

"C"-Funktionsbausteine ordnen Operationen statische Daten zu. Sie sind Ergänzungen der "C"-Funktionen, indem sie die Verarbeitung statischer Objekte ermöglichen. Sie werden gewöhnlich benutzt, um die Standardkapazitäten der **ST**- und **FBS**-Sprachen zu erweitern. Im Gegensatz zu Funktionen, die lediglich Werte verarbeiten, können Funktionsbausteine statische Daten verarbeiten, was bedeutet, daß ein Funktionsbaustein-Algorithmus die zeitliche Entwicklung seiner Daten verwalten kann.

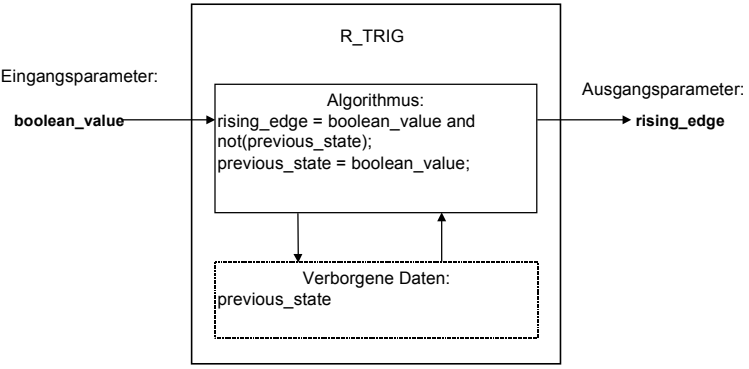
Die Funktionsbausteine werden in der "C"-Sprache geschrieben, kompiliert und mit dem ISaGRAF Kernel verbunden. Der neue Kernel muß auf der ISaGRAF Zielsteuerung installiert werden, bevor die neuen Funktionsbausteine in den ISaGRAF Projekten benutzt werden können. Da diese Funktionsbausteine nicht in den ISaGRAF Simulator integriert werden können, müssen die ISaGRAF Anwendungen simuliert werden, **bevor** die nicht-standard Funktionsbausteine hinzugefügt werden.

Warnung: Funktionsbausteine sind **synchrone** Operationen, die vom ISaGRAF Kernel während des Ausführungszylus einer Anwendung aktiviert werden. Die für die Ausführung eines Aktivierungs- oder Lese-Service eines Funktionsbausteins benötigte Zeit ist in der **Zykluszeit** der ISaGRAF Anwendung enthalten. Der Benutzer sollte sich vergewissern, daß

der Funktionsbaustein keine "Warteoperationen" enthält, so daß der ISaGRAF Zyklus die maximale Zykluszeit nicht überschreitet.

Deklaration der Funktionsbaustein-Instancen

Funktionsbausteine sind Objekte, die Operationen und statische Daten miteinander kombinieren. Es folgt das Beispiel eines "**R_TRIG**" Funktionsbausteins, der die steigende Flanke eines booleschen Ausdrucks erkennt. Hier die funktionelle Beschreibung des Bausteins:



Die verborgene statische Variable "**previous_state**" wird für die Flankenberechnung benötigt. Diese Variable muß jedesmal, wenn der Funktionsbaustein " **TRIG**" in der Anwendung benutzt wird, unterschiedlich sein. Die in der ST-Sprache benutzten Instancen des Funktionsbausteins müssen im Datenverzeichnis deklariert werden. Da Funktionsbausteine verborgene interne Daten besitzen, muß jede Kopie (Instance) eines Funktionsbausteins mit einem einmaligen Namen gekennzeichnet werden. Der Typ eines Funktionsbausteins wird mit dem Bibliotheksmanager bestimmt, die Instancen werden mit dem Datenverzeichnis-Editor benannt.

Die in der FBS-Sprache benutzten Funktionbausteine brauchen nicht dekariert zu werden, da der ISaGRAF FBS-Editor die Instancen der benutzen Funktionsbausteine automatisch deklariert. Funktionsbaustein-Instancen, die automatisch vom FBS-Editor deklariert werden, sind stets **LOKAL** zum bearbeiteten Programm.

Hinzufügen eines Funktionsbaustein in die ISaGRAF Bibliothek

Um der ISaGRAF Bibliothek der Workstation einen neuen "C"-Funktionsbaustein hinzuzufügen, benutzt man den ISaGRAF Bibliotheksmanager. Öffnen Sie die Bibliothek der Funktionsbausteine und leiten Sie den Befehl "**Neu**" im "**Datei**"-Menü ein. Nach der Erstellung eines neuen Funktionsbausteins muß sein **Datenblatt** abgefaßt werden. Der Block des "C"-Quellcodes für den neuen Funktionsbaustein wird automatisch vom ISaGRAF Bibliotheksmanager generiert. Der Befehl "**Parameter**" im "**Datei**"-Menü wird benutzt, um die Aufruf- und Returnparameter des neuen Funktionsbausteins zu definieren.

Gebrauch eines "C"-Funktionsbausteins im ISaGRAF Projekt

Beliebige integrierte "C"-Funktionsbausteine können in den Programmen eines ISaGRAF Projekts benutzt werden. "C"-Funktionsbausteine können von den **ST**- und **FBS**-Sprachen aufgerufen werden.

Der Aufruf eines "C"-Funktionsbausteins von der **ST**-Sprache folgt den Funktionsbausteinaufrufkonventionen dieser Sprache. Die Aufrufparameter des Funktionsbausteins werden hinter den Namen der Funktion in Klammern geschrieben und durch Kommas getrennt. Auf die Returnparameter wird nacheinander zugegriffen. Jeder Returnparameter wird mit einem Namen gekennzeichnet, der sich aus dem Namen der Baustein-Instance und dem Namen des Parameters zusammensetzt. Die einzelnen Komponenten werden mit einem Punkt getrennt. Der folgende Name:

FBINSTNAME.parname

wird benutzt, um den Returnparameter namens "**parname**" der Funktionsbaustein-Instance "**FBINSTNAME**" darzustellen.

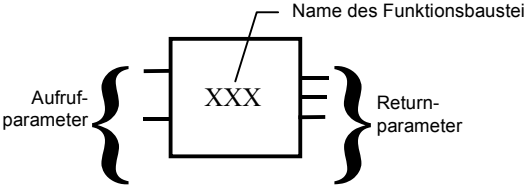
Die in der ST-Sprache benutzten Instancen eines Funktionsbausteins müssen im Datenverzeichnis deklariert werden. Jede Kopie (Instance) eines Funktionsbausteins muß mit einem einmaligen Namen gekennzeichnet werden. Es folgt das Beispiel einer Instancen-Deklaration im ISaGRAF Datenverzeichnis:

Instance:	TRIG1	Typ:	R_TRIG
	TRIG2		R_TRIG

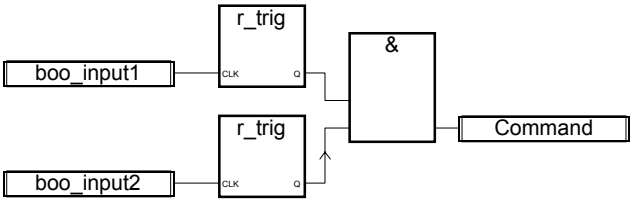
Hier ein Beispiel, in dem diese deklarierten Instancen in einem ST-Programm benutzt werden:

```
TRIG1 (boo_input1);
TRIG2 (boo_input2);
Command := (TRIG1.Q & TRIG2.Q);
```

Ein **FBS**-Programm kann einen beliebigen "C"-Funktionsbaustein aufrufen. Ein Funktionsbaustein wird wie eine Standard-Funktionsbox benutzt. Seine Aufrufparameter sind mit der linken Seite der Funktionsbox verbunden, seine Returnparameter mit der rechten Seite der Box. Hier das Standardformat einer Funktionsbox:

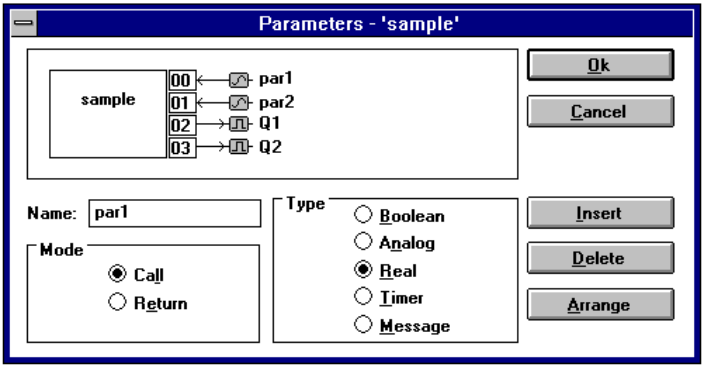


Die in der FBS-Sprache benutzten Funktionbausteine brauchen nicht dekariert zu werden, da der ISaGRAF FBS-Editor die Instancen der benutzen Funktionsbausteine automatisch deklariert. Funktionsbaustein-Instancen, die automatisch vom FBS-Editor deklariert werden, sind stets **LOKAL** zum bearbeiteten Programm. Es folgt das vorherige Beispiel in der FBS-Sprache programmiert:



Schnittstellendefinition eines "C"-Funktionsbausteins

Der Befehl "**Parameter**" im "**Datei**"-Menü öffnet ein Dialogfeld, in dem die Aufruf- und Returnparameter eines neuen Funktionsbausteins definiert werden. Ein Funktionsbaustein besitzt bis zu **32** Parameter, die sich beliebig aus Aufruf- und Returnparametern zusammensetzen können. Im Gegensatz zu den "C"-Funktionen können Funktionsbausteine mehrere Returnparameter besitzen. Das folgende Dialogfeld dient der Parameterdefinition eines "C"-Funktionsbausteins:



Im oberen Teil des Fensters erscheinen die Parameter des "C"-Funktionsbaustein in der Reihenfolge des Funktionsaufruf-Prototyps: zuerst die Aufrufparameter und zuletzt die Returnparameter. Im unteren Teil des Fensters erscheint eine detaillierte Beschreibung des derzeit in der Liste ausgewählten Parameters:

- Name des Parameters
- Richtung (Aufruf/Return) des Parameters
- Typ des Parameters

Sämtliche ISaGRAF Datentypen können benutzt werden: boolesch, ganzzahlig analog, real analog, Timer oder Zeichenkette. Zwischen ganzzahlig analog und real analog wird unterschieden.

Es folgt die Entsprechung zwischen ISaGRAF Typen und "C"-Typen:

BOOLESCH	unsigned long	vorzeichenloses 32 Bit-Wort: 1=true / 0=false

ANALOG	long	vorzeichenbehaftetes ganzzahliges 32 Bit-Wort
REAL	float	Gleitpunktwert mit einfacher Präzision
TIMER	unsigned long	vorzeichenloses ganzzahliges 32 Bit-Wort (Einheit = 1 Millisekunde)
ZEICHENKETTE	char *	Zeichenkette

Wenn ein Zeichenkettenwert an einen "C"-Funktionsbaustein gegeben wird, kann er keine Nullzeichen enthalten, denn die an den "C"-Code gegebene Zeichenkette endet in null. Vergessen Sie nicht, daß die Returnparameter zuletzt in der Liste erscheinen müssen. Bei der Parameterbenennung müssen die folgenden Regeln beachtet werden:

- Der Name darf nicht länger als 16 Zeichen sein.
- Das erste Zeichen muß ein Buchstabe sein.
- Die weiteren Zeichen müssen Buchstaben, Zahlen oder Unterstrichzeichen sein.
- Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Ein Parametername darf nicht von mehreren Parametern eines Funktionsbausteins benutzt werden. Ein Aufrufparameter darf nicht den gleichen Namen wie ein Returnparameter haben. Gleiche Namen können hingegen für Parameter **unterschiedlicher** Funktionsbausteine benutzt werden. Parameternamen werden benutzt, um die betreffenden Parameter im "C"-Quellcode zu identifizieren.

Mit dem Befehl "**Einfügen**" kann ein neuer Parameter vor dem ausgewählten Parameter eingefügt werden. Der Befehl "**Löschen**" löscht den ausgewählten Parameter. Der Befehl "**Arrangieren**" sortiert die Parameter automatisch und plaziert die Returnparameter an das Ende der Liste. Klicken Sie auf die Schaltfläche "**OK**", um die Definitionen der Funktionsbausteinschnittstelle zu speichern und das Dialogfeld zu schließen. Klicken Sie auf "**Abbrechen**", um das Dialogfeld zu schließen, ohne die Modifikationen zu speichern.

"C"-Schnittstelle eines Funktionsbausteins

Die Schnittstelle eines Funktionsbausteins ist abhängig von der Definition seiner Parameter.
Die Aufrufparameter werden in eine Struktur gegeben. Diese Struktur wird in der Datei "**GRFB0nnn.H**" definiert, in der "**nnn**" die logische Nummer des Funktionsbausteins in der ISaGRAF Bibliothek ist. Die Returnparameter werden durch logische Nummern dargestellt, die ebenfalls in der Datei "**GRFB0nnn.H**" definiert werden.

Hier ein Beispiel einer "C"-Schnittstelle für den Funktionsbaustein "**LIM_ALRM**" (Grenzalarm):

```
/* Funktionsbaustein-Schnittstelle - Name: sample */

/* ISaGRAF Standard-Datentypen */

typedef          long     T_BOO;
typedef          long     T_ANA;
typedef          float    T_REAL;
typedef          long     T_TMR;
```

```
typedef                         char    *T_MSG;

/* Struktur der Aufrufparameter */

typedef struct {
        /* AUFRUF   */          T_BOO _par1;
        /* AUFRUF   */          T_BOO _par2;
} str_arg;

/* Zugriff auf die Felder der Struktur str_arg */

#define                         PAR1    (arg->_par1)
#define                         PAR2    (arg->_par2)

/* Rückgabe der logischen Parameternummern */

#define                         FBLPNO_Q1       0
#define                         FBLPNO_Q2       1

/* Dateiende */
```

Es folgen die Entsprechungen zwischen den ISaGRAF Typen und den "C"-Typen. Die ISaGRAF Typen werden in der Definitionsdatei des Funktionsbausteins als "C"-Typen definiert.

boolesch	T_BOO	long (32 Bits)
analog	T_ANA	long
real	T_REAL	float (32 Bits - einfache Präzision)
Timer	T_TMR	long
Zeichenkette	T_MSG	char * (32 Bits - Zeichenpointer)

Jedes Feld der "**str_arg**" Struktur entspricht einem Parameter des Funktionsbausteins. Die Parameter erscheinen in der Struktur in der gleichen Reihenfolge wie in der Definition des Funktionsbausteins. Ein Bezeichner in Großbuchstaben wird definiert, um den direkten Zugriff auf einen Parameter der an die "C"-Implementierung des Aktivierungs-Service des Funktionsbausteins gegebenen Struktur herzustellen. Die Namen der Bezeichner sind jene, welche mit dem ISaGRAF Bibliotheksmanager bei der Definition des Funktionsbausteins eingegeben wurden.

Die Returnparameter erscheinen in der gleichen Reihenfolge wie in der Definition des Funktionsbausteins. Die logische Nummer des ersten Returnparameters ist immer **0**.

In der "C"-Quellcodeprogrammierung sollten Bezeichner anstatt numerischer Werte benutzt werden, um die Returnparameter darzustellen. Dadurch wird sichergestellt, daß die Quelldatei nach einer Modifizierung der Schnittstellendefinition problemlos neu kompiliert werden kann.

Die "C"-Definitionsdatei wird jedesmal aktualisiert, wenn die Funktionsbausteinschnittstelle im ISaGRAF Bibliotheksmanager modifiziert wird. Dies sichert die Konsistenz zwischen der Implementierung des Funktionsbausteins und seiner Verwendung in den Programmen der ISaGRAF Anwendungen.

Quellcode

Die "C"-Implementierung eines Funktionsbausteins besteht aus drei verschiedenen Eingangspunkten:

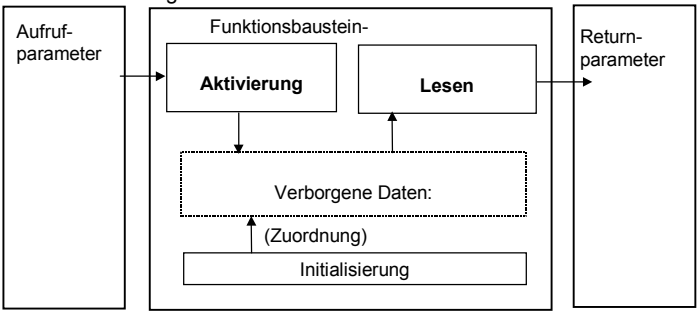
- ❑ Initialisierungsservice
- ❑ Aktivierungsservice - Verarbeitung der Aufrufparameter
- ❑ Leseservice - Zugriff auf die Returnparameter

Für jede Instance eines Funktionsbausteins wird derselbe Code benutzt (nicht kopiert). Jeder Instance wird eine statische Datenstruktur zugeordnet. Es ist nicht möglich, mit der ISaGRAF Programmierung direkt auf solche Daten zuzugreifen, die "verborgene Variablen" der Funktionsbaustein-Instance enthalten.

Der "Aktivierungsservice" wird einmal in jedem Zielsystemzyklus für jede Instance eines jeden benutzten Bausteins aufgerufen. Er verarbeitet die Aufrufparameter und aktualisiert die zugeordneten Daten. Er ist der "Hauptalgorithmus" des Funktionsbausteins.

Der "Leseservice" wird vom ISaGRAF Kernel aufgerufen, um den aktuellen Wert eines Returnparameters einer Instance zu lesen. In diesem Service erfolgt keine spezielle Berechnung. Er dient lediglich der Übertragung zwischen den verborgenen Daten und der ISaGRAF Anwendung.

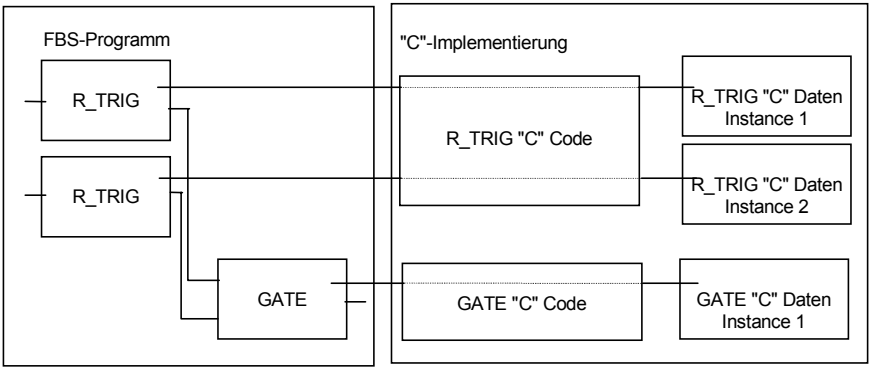
Funktionelles Diagramm:



- **Statische Daten eines Funktionsbausteins**

Funktionsbausteine ordnen Operationen statische Daten zu. Jeder Instance eines Funktionsbausteins wird eine Datenstruktur zugeordnet. Jedesmal wenn ein Funktionsbaustein in einem ST- oder FBS-Programm benutzt wird, entspricht er einer Instance und einer Datenstruktur. Das folgende Beispiel veranschaulicht die Entsprechungen

zwischen "C"-Datenstrukturen und den in einem FBS-Programm benutzten Funktionsbaustein-Instancen:



Der für jede Datenstruktur jeder Instance erforderliche Speicherplatz wird vom ISaGRAF System bei Anwendungsstart zugeordnet. Ein Pointer zu der zugeordneten Datenstruktur wird an die "Aktivierungs-" und "Leseservice" gegeben.

Der Standardblock des "C"-Quellcodes für die Typdefinition der Datenstruktur wird automatisch vom ISaGRAF Bibliotheksmanager generiert. Der Datenstrukturtyp heißt immer "**str_data**". Dieser Name darf nicht geändert werden, damit die Kompatibilität mit den Service-Vorsätzen erhalten bleibt. Die verborgenen Daten enthalten im allgemeinen interne Variablen und ein Bild der Returnparameter. Der "Leseservice" eines Funktionsbausteins wird lediglich benutzt, um auf die Returnparameter zuzugreifen und sollte nicht für andere Operationen verwendet werden.

- **Der Initialisierungsservice**

Der "Initialisierungsservice" wird vom ISaGRAF Kernel beim Anwendungsstart aufgerufen. Er gibt dem "C"-Programmierer die Möglichkeit, das System zu beauftragen, einer Instance Speicherplatz zuzuordnen. Hier die Standardprogrammierung des Initialisierungsservice:

```
uint16 FBINIT_xxx (uint16 hinstance)
/* "xxx" ist der Name des Funktionsbausteins */
{
  return (sizeof (str_data));
}
```

Das Argument "**hinstance**" ist die logische Nummer der Instance. Es ist für interne ISaGRAF Operationen reserviert und sollte nicht in der Programmierung des Service benutzt werden. Der Initialisierungsservice liefert die Anzahl der Speicherbytes, die für die Daten **einer** Instance benötigt werden. Der benötigte Speicherplatz (Rückgabewert) darf **64** Kbytes nicht überschreiten. In diesem Service dürfen keine anderen Operationen ausgeführt werden. Der "C"-Quellcode des Service wird automatisch bei der Erstellung des Funktionsbausteins vom ISaGRAF Bibliotheksmanager generiert.

- **Der Aktivierungsservice**

Der "Aktivierungsservice" wird bei jedem Zielsystemzyklus aufgerufen, und zwar für jede in der Anwendung benutzte Funktionsbaustein-Instance. Dieser Service verarbeitet die Aufrufparameter und führt den Hauptalgorithmus des Funktionsbausteins aus, um die verborgenen statischen Daten und die Werte der Returnparameter zu aktualisieren. Hier der Standardblock des Aktivierungsservice:

```
void FBACT_xxx (
uint16 hinstance,              /* "xxx"ist der Name des F-Bausteins */
                               /* logische Nummer der Instance */
str_data *data,                /* data: Pointer z. Datenstruktur der Instance */
str_arg *arg                   /* Pointer z. Struktur der Aufrufparameter */
)
{
}
```

Das Argument "**hinstance**" ist die logische Nummer der Instance. Es ist für interne ISaGRAF Operationen reserviert und sollte nicht in der Programmierung des Service benutzt werden. Das Argument "**data**" ist ein Pointer zu der der Instance zugeordneten Datenstruktur. Das Argument "**arg**" ist ein Pointer zu der Struktur, welche die Werte der Aufrufparameter enthält. Der Programmierer sollte die im "C"-Vorsatz des Funktionsbausteins definierten Bezeichner benutzen, um auf die Felder der "**arg**" Struktur zuzugreifen.

Der "Aktivierungsalgorithmus" verarbeitet die (in der "**arg**" Struktur gespeicherten) Aufrufparameter und aktualisiert die Felder der "**data**" Struktur. Das folgende Beispiel zeigt den "Aktivierungsservice" des Funktionsbausteins **TRIG** (steigende Flankenerkennung):

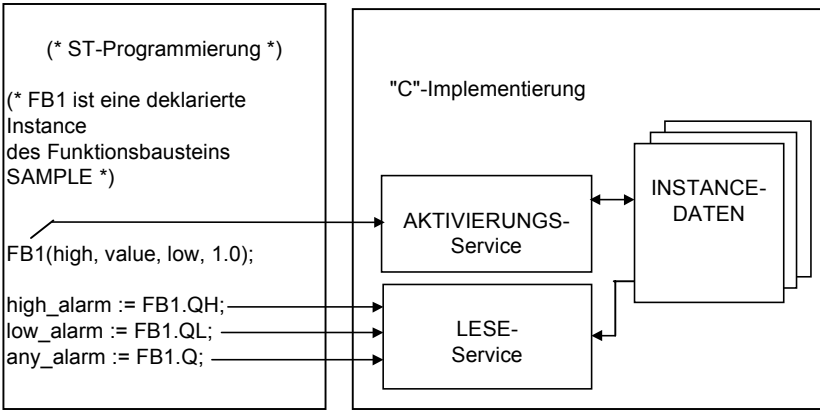
```
/* im "C"-Vorsatz des Funktionsbausteins gespeicherte Definitionen*/

typedef struct {               /* Aufrufparameter */
        T_BOO  _clk;           /* Trigger-Eingang */
} str_arg;

#define CLK      (arg->_clk)

/* Datenstruktur der Funktionsbaustein-Instance */

typedef struct {
        T_BOO prev_state;      /* vorheriger Status des Trigger-Eingangs */
        T_BOO edge_detect;     /* Flankenwert: Bild des Returnparameters */
} str_data;

/* Aktivierungsservice */

void FBACT_trig (uint16 hinstance, str_data *data, str_arg *arg)
{
```

```
        data->edge_detect = (T_BOO)(CLK && !data->prev_state);
        data->prev_state = CLK;  /* Aufrufparameter */
}
```

Der "C"Quellcodeblock dieses Service wird automatisch bei der Erstellung des Funktionsbausteins vom ISaGRAF Bibliotheksmanager generiert.

- **Lesen der Returnparameter**

Der "Leseservice" wird jedesmal aufgerufen, wenn ein Returnparameter einer Funktionsbaustein-Instance in einem ST- oder FBS-Programm referenziert ist. Er wird benutzt, um den Wert **eines** Returnparameters einzuholen. Das folgende Beispiel zeigt die "Leseaufrufe", die bei der Ausführung eines ST-Programms erfolgen:



Da der "Leseservice" mehrmals in einem Zyklus für denselben Returnparameter einer Funktionsbaustein-Instance aufgerufen werden kann, dürfen keine speziellen Berechnungen in diesem Service erfolgen. Der Leseservice führt lediglich Übertragungen zwischen den verborgenen Daten und der ISaGRAF Anwendung aus. Es folgt der Standardblock des Leseservice:

```
/* Operation für die Kopie eines Returnparameterwerts */

#define BOO_VALUE       ((T_BOO *)value)
#define ANA_VALUE       ((T_ANA *)value)
#define REAL_VALUE      ((T_REAL *)value)
#define TMR_VALUE       ((T_TMR *)value)
#define MSG_VALUE       ((T_MSG *)value)

/* Returnparameter-Leseservice: wird für jeden Returnparameter aufgerufen */
```

```
void FBREAD_xxx (               /* "xxx" ist der Name des Funktionsbausteins
*/
uint16 hinstance,               /* logische Nummer der Instance */
str_data *data,                 /* Pointer zur Datenstruktur der Instance */
uint16 parno,                   /* logische Nummer des gelesenen Parameters
*/
void *value)                    /* Puffer, in dem der Parameterwert kopiert
wird */
{
        switch (parno) {
            case FBLPNO_XX: /* ... */ break;
            case FBLPNO_YY: /* ... */ break;
             /* .... */
        }
}
```

Das Argument "**hinstance**" ist die logische Nummer der Instance. Es ist für interne ISaGRAF Operationen reserviert und sollte nicht in der Programmierung des Service benutzt werden. Das Argument "**data**" ist ein Pointer zu der der Instance zugeordneten Datenstruktur.

Das Argument "**parno**" ist die logische Nummer des Returnparameters, dessen Wert benötigt wird. Benutzen Sie die im "C"-Vorsatz des Funktionsbausteins definierten Bezeichner, um die Returnparameter zu kennzeichnen. Solche Bezeichner beginnen mit dem Vorzeichen "**FBLPNO_**". Das Argument "**value**" ist ein Pointer zu einem Puffer, in dem der aktuelle Wert des zugegriffenen Returnparameters kopiert wird. Der durch dieses Argument pointierte Datentyp ist abhängig vom ISaGRAF Typ des Returnparameters. Die folgende Tabelle liefert die Beziehungen zwischen ISaGRAF Typen und Puffer-"C"-Datentypen:

boolesch	long	vorzeichenloses 32 Bit-Wort (0=false / 1=true)
analog	long	vorzeichenbehaftetes 32 Bit-Wort
real	float	32 Bit-Gleitpunktwert mit einfacher Präzision
Timer	long	vorzeichenloses 32 Bit-Wort (Einheit = 1ms)
Zeichenkette	char *	Zeichenfeld

Die folgenden Makros werden benutzt, um auf den Kopie-Puffer zuzugreifen, entsprechend des Typs des zugegriffenen Returnparameters:

```
#define     BOO_VALUE       ((T_BOO *)value)
#define     ANA_VALUE       ((T_ANA *)value)
#define     REAL_VALUE      ((T_REAL *)value)
#define     TMR_VALUE       ((T_TMR *)value)
#define     MSG_VALUE       ((T_MSG *)value)
```

Gewöhnlich werden die folgenden Operationen benutzt, um den Parameterwert in den ISaGRAF Puffer zu kopieren:

```
/* für einen booleschen Parameter: */
  *BOO_VALUE = parameter_value;
/* für einen ganzzahligen analogen Parameter: */
  *ANA_VALUE = parameter_value;
/* für einen realen analogen Parameter: */
  *REAL_VALUE = parameter_value;
/* für einen Timer-Parameter: */
  *TMR_VALUE = parameter_value;
/* für einen Zeichenketten-Parameter: */
  strcpy (*MSG_VALUE, parameter_value);
```

Der "C"-Quellcodeblock dieses Service wird bei der Erstellung des Funktionsbausteins automatisch vom ISaGRAF Bibliotheksmanager generiert.

- **Beispiel einer "C"-Quelldatei**

Es folgt der Standardblock einer "C"-Funktionsbaustein-Implementierung:

```
/* Funktionsbaustein (xxx ist der Name des Funktionsbausteins) */

#include <tasy0def.h>
#include <grfb0nnn.h>   /* nnn ist die Nummer des F-Bausteins in der Bibliothek
*/

/* Struktur verborgener Daten für jede Instance des Bausteins */
typedef struct {
        /* Felddefinition */
} str_data;

/* Initialisierungsservice: liefert die Größe der erforderlichen verborgenen Daten */
word FBINIT_xxx (uint16 hinstance)
{
        return (sizeof (str_data));
}

/* Aktivierungsservice: verarbeitet die Aufrufparameter */
void FBACT_xxx (uint16 hinstance, str_data *data, str_arg *arg)
{
        /* ... */
}

/* Operation für die Kopie des Wertes eines Returnparameters */
#define BOO_VALUE   ((T_BOO *)value)
```

```
#define ANA_VALUE   ((T_ANA *)value)
#define REAL_VALUE  ((T_REAL *)value)
#define TMR_VALUE   ((T_TMR *)value)
#define MSG_VALUE   ((T_MSG *)value)

/* Returnparameter-Leseservice: wird für jeden Returnparameter aufgerufen */
void FBREAD_xxx (uint16 hinstance, str_data *data, uint16 parno, void *value)
{
        switch(parno)
        {
          case FBLPNO_XX: *???_VALUE = ...; break;
          case FBLPNO_YY: *???_VALUE = ...; break;
        ....

}
```

/* Die folgende Funktion wird für die Initialisierung des Funktionsbausteins und die Deklaration seiner Implementierung benutzt. Sie stellt die Verbindung mit dem ISaGRAF Kernel unter Anwendung des Funktionsbausteinnamens her. Dieser Service wird vollständig vom ISaGRAF Bibliotheksmanager generiert. */

```
ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
{
        strcpy (name, "XXX");
        *initproc = (IBP)FBINIT_xxx;
        *readproc = (RBP)FBREAD_xxx;
        return ((ABP)FBACT_xxx);
}

/* Dateiende */
```

Die "**TASY0DEF.H**" Include-Datei des ISaGRAF Kernels wird für die system-abhängigen Definitionen gebraucht. Ferner enthält sie die Definition von Datentypen, die Far-Pointer zu den implementierten Services darstellen.

Verbindungen zwischen Projekten und "C"-Implementierung

Die logische Verbindung zwischen der Implementierung eines "C"-Funktionsbausteins und seiner Benutzung in den Programmen eines ISaGRAF Projekts erfolgt mit dem Namen des Funktionsbausteins. Ein "Deklarationsservice" wird dem "C"-Quellcode des Funktionsbausteins beigefügt. Dieser Service wird nur ein einziges Mal beim Start der Anwendung aufgerufen und teilt dem ISaGRAF Kernel den Namen des "C"-Funktionsbausteins mit, welcher den implementierten Services entspricht. Hier das Standardformat eines solchen Deklarationsservice:

```
ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
{
        strcpy (name, "XXX");       /* Name des Funktionsbausteins */
        *initproc = (IBP)FBINIT_xxx;            /* Initialisierungsservice */
        *readproc = (RBP)FBREAD_xxx;            /* Leseservice */
        return ((ABP)FBACT_xxx);                /* Aktivierungsservice */
}
/* xxx ist der Name des Funktionsbausteins */
```

Der Name des Funktionsbausteins wird **großgeschrieben**, wenn er in der **strcpy** Anweisung benutzt wird. Er wird kleingeschrieben, wenn er für den Namen der implementierten Services und im Namen des Deklarationsservice benutzt wird.

Wenn die Vorzeichen "**FBACT_**", "**FBINIT_**", "**FBREAD_**" und "**fbldef_**" für die implementierten Services und den Definitionsservice benutzt werden, kann ein Funktionsbaustein mit einem reservierten Schlüsselwort der "C"-Sprache oder mit dem Namen einer Funktion der ISaGRAF "C"-Bibliotheken benannt werden. Dem Deklarationsservice sollten keine weiteren Anweisungen hinzugefügt werden.

Der Deklarationsservice wird für alle integrierten "C"-Funktionsbausteine aufgerufen, selbst wenn diese nicht in den Programmen der ISaGRAF Anwendung benutzt werden. Der ISaGRAF Kernel generiert einen fatalen Fehler, wenn ein in der Anwendung benutzter "C"-Funktionsbaustein nicht im Kernel integriert ist.

Bevor neue Funktionsbausteine mit dem Kernel verbunden werden können, muß der Benutzer eine neue "C"-Quelldatei namens "**GRFB0LIB.C**" schreiben und (mit den festgehaltenen Funktionsbausteinen) in die Dateiliste für die Verbindung einfügen. "**GRFB0LIB.C**" enthält lediglich ein Feld mit Deklarationsfunktionen. Dieses Feld wird bei der Anwendungsinitialisierung gelesen, um eine dynamische Verbindung mit den in "C" geschriebenen Funktionsbausteinen herzustellen.
Hier ein Beispiel:

```
/* Datei: grfb0lib.c - implementierte Funktionsbausteine */

#include <tasy0def.h>

extern ABP fbldef_fb1(char *name, IBP *init, RBP *read);
extern ABP fbldef_fb2(char *name, IBP *init, RBP *read);

FBL_LIST FBLDEF[ ] = {
        fbldef_fb1,
        fbldef_fb2,

NULL };

/* Dateiende */
```

Das Feld **FBLDEF** muß in einem NULL Pointer enden, sonst könnten schwerwiegende Probleme auftreten. Wenn das Feld nicht definiert wird, treten ungelöste Referenzen auf, wenn der neue ISaGRAF Kernel verbunden wird.

Diese Datei ermöglicht den Aufbau eines neuen Kernels, einschließlich aller bestehenden Funktionsbausteine. Es ist ebenfalls möglich, einen projekt-dedizierten Kernel zu erstellen, indem nur Funktionsbausteine eines bestimmten Projekts in das **FBLDEF** Feld aufgenommen werden. Die Datei "**GRFB0LIB.C**" wird automatisch vom ISaGRAF Codegenerator generiert, wenn der Anwendungscode entwickelt wird. Die Datei wird im ISaGRAF Projekt verzeichnis gespeichert und enthält lediglich die Funktionsbausteine des Projekts.

Begrenzungen

Die ISaGRAF Bibliothek kann bis zu **255** "C"-Funktionsbausteine enthalten. In einer Funktion können beliebige Operationen verarbeitet werden. Alle Funktionsbausteintypen können bis zu **255** mal in einem Projekt kopiert (instanziiert) werden.

Man sollte nicht vergessen, daß die Funktionsbaustein-Services im ISaGRAF Zyklus **synchron** aufgerufen werden, so daß die Ausführung eines Funktionsbausteins einen direkten Einfluß auf die Zykluszeit nimmt.

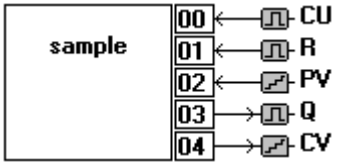
Abgeschlossenes Beispiel

Es folgt die komplette Programmierung eines "**sample**" Funktionsbausteins, der ein Inkrementalzähler ist.

Hier das technische Datenblatt des Funktionsbausteins:

Name:	SAMPLE
Beschreibung:	Inkrementalzähler
Erstellungsdatum:	1.2.1994
Autor:	CJ International
Aufruf:	CU : Zähleingang
	R : Befehl rücksetzen
	PV : programmierter Maximalwert
Return:	Q : Maximalwerterkennung
	CV : Zählergebnis
Prototyp:	SAMPLE (count, reset_command, maximum_value);
	max_detect := SAMPLE.Q;
	count_result := SAMPLE.CV;

Es folgt die Schnittstelle des Funktionsbausteins:



Es folgt der "C"-Quellcode-Vorsatz des Funktionsbausteins:

```
/* Funktionsbaustein-Schnittstelle - Name: SAMPLE */

/* Definition der ISaGRAF Standarddatentypen */

typedef long  T_BOO;
typedef long  T_ANA;
typedef float T_REAL;
typedef long  T_TMR;
typedef char  *T_MSG;

/* Definition der Struktur der Aufrufparameter */

typedef struct {
  T_BOO  _cu;
  T_BOO  _r;
  T_ANA  _pv;
} str_arg;

/* Bezeichner für den Zugriff auf die Aufrufparameter */

#define CU                  (arg->_cu)
#define R                   (arg->_r)
#define PV                  (arg->_pv)

/* logische Nummerierung der Returnparameter */

#define FBLPNO_Q            0
#define FBLPNO_CV           1

/* Dateiende */
```

Es folgt der "C"-Quellcode des Funktionsbausteins. Ausschließlich die fettgedruckten Zeilen wurden manuell vom Programmierer eingegeben.

```
/* Funktionsbaustein - Name: SAMPLE */

#include <tasy0def.h>          /* erforderlich für die Datentypdefinition */
#include <grfb0255.h>          /* C-Quellcodevorsatz des Funktionsbausteins
*/

/* Definition der Struktur, die die Daten für eine Instance enthält */

typedef struct {
        T_BOO overflow;        /* true:Zählwert >= programmierter Wert */
        T_ANA value;           /* aktueller Zählwert */
} str_data;

/* Initialisierungsservice: fordert Speicherplatz für Instance-Daten */

word FBINIT_sample (uint16 hinstance)
{
        return (sizeof (str_data));
}

/* Aktivierungsservice: Zähl-Algorithmus */

void FBACT_sample (uint16 hinstance, str_data *data, str_arg *arg)
{
        if (R) data->value = 0;
        else if (CU && data->value < PV) (data->value)++;
        data->overflow = (data->value >= PV) ? (T_BOO)1 : (T_BOO)0;
}

/* Operation für die Kopie der Parameter in den ISaGRAF Puffer */

#define BOO_VALUE   ((T_BOO *)value)
#define ANA_VALUE   ((T_ANA *)value)
#define REAL_VALUE  ((T_REAL *)value)
#define TMR_VALUE   ((T_TMR *)value)
#define MSG_VALUE   ((T_MSG *)value)

/* Leseservice: holt den Wert eines Returnparameters ein */

void FBREAD_sample (uint16 hinstance, str_data *data, uint16 parno, void
*value)
```

```
{
        switch (parno) {
          case FBLPNO_Q   : *BOO_VALUE = data->overflow; break;
          case FBLPNO_CV : *ANA_VALUE = data->value; break;
        }
}

/* Deklarationsservice für die dynamische Verbindung mit dem ISaGRAF Kernel
*/

ABP fbldef_sample (char *name, IBP *initproc, RBP *readproc)
{
        strcpy (name, "SAMPLE");
        *initproc = (IBP)FBINIT_sample;
        *readproc = (RBP)FBREAD_sample;
        return ((ABP)FBACT_sample);
}

/* Dateiende */
```

C.7.5. Kompilierungs- und Integrationstechniken

Obwohl die ISaGRAF Workstation über keinen "C"-Compiler oder Linker verfügt, ist es vielleicht nützlich, die Grundtechniken für den Gebrauch der mit dem ISaGRAF Bibliotheksmanager erstellten Dateien und deren Bearbeitung mit solchen Werkzeugen in diesem Kapitel näher zu erläutern.

"C"-Quelldateien

Die "C"-Quelldateien der Umrechnungen, Funktionen und Funktionsbausteine werden vom ISaGRAF Bibliotheksmanager in den Verzeichnissen **ISAWIN\LIB\DEFS** und **ISAWIN\LIB\SRC** gespeichert.

Der Name einer Quelldatei ergibt sich aus der Nummer der betreffenden Umrechnung, Funktion oder des Funktionsbausteins in der ISaGRAF Bibliothek.

Die folgenden Dateinamen werden benutzt:

\isawin\lib\defs\TACN0DEF.H	Definitionsdatei f. Umrechnungsfunktionen
\isawin\lib\src\GRCN0nnn.H	Quelldatei einer Umrechnungsfunktion
\isawin\lib\defs\GRUS0nnn.H	Definitionsdatei einer Funktion
\isawin\lib\src\GRUS0nnn.C	Quelldatei einer Funktion
\isawin\lib\defs\GRFB0nnn.H	Definitionsdatei eines Funktionsbausteins
\isawin\lib\src\GRFB0nnn.C	Quelldatei eines Funktionsbausteins

(**nnn** ist die Nummer der Umrechnung, Funktion oder des Funktionsbausteins)

Warnung: Wenn Bibliothekselemente umbenannt oder kopiert werden, werden ihre Text- und Programmierungszeilen nicht vom ISaGRAF Bibliotheksmanager mit dem neuen Elementnamen und der neuen logischen Nummer aktualisiert. Sie müssen manuell in der "C"-Quelldatei aktualisiert werden.

Die Datei **\ISAWIN\LIB\USPNUMS** liefert die Beziehung zwischen den Namen und logischen Nummern der "C"-Funktionen der ISaGRAF Bibliothek. Hier ein Beispiel einer solchen Datei:

```
1     funct_A
10    funct_B
16    funct_C
```

Die Datei **\ISAWIN\LIB\FBLNUMS** liefert die Beziehung zwischen den Namen und logischen Nummern der "C"-Funktionsbausteine der ISaGRAF Bibliothek. Hier ein Beispiel einer solchen Datei::

```
0     fbl_A
1     fbl_B
2     fbl_C
```

Die Datei **\ISAWIN\LIB\CNVNUMS** liefert die Beziehung zwischen den Namen und logischen Nummern der Umrechnungsfunktionen der ISaGRAF Bibliothek. Hier zum Beipiel der Inhalt dieser Datei für die Umrechnungen der Standardbibliothek:

```
0     SCALE
1     BCD
```

Diese Dateien werden automatisch vom ISaGRAF Bibliotheksmanager aktualisiert, wenn eine Umrechnung, eine Funktion oder ein Funktionsbaustein erstellt, umbenannt, kopiert oder gelöscht wird. Bei der Codeerstellung einer Anwendung werden die folgenden Dateien automatisch vom ISaGRAF Codegenerator generiert:

\isawin\apl\ppp\GRCN0LIB.C	Deklaration als ein Feld aller im Projekt benutzten Umrechnungsfunktionen.
\isawin\apl\ppp\GRUS0LIB.C	Deklaration als ein Feld aller im Projekt benutzten Funktionen.
\isawin\apl\ppp\GRFB0LIB.C	Deklaration als ein Feld aller im Projekt benutzten Funktionsbausteine.

(**ppp** ist der Name des ISaGRAF Projekts)

Diese Dateien können bei Link-Operationen benutzt werden, um einen neuen ISaGRAF Kernel aufzubauen, der ausschließlich die im Projekt benutzten Umrechnungen, Funktionen und Funktionsbausteine enthält.

Laden der Quelldateien auf eine Datenstation

Die vom ISaGRAF Bibliotheksmanager erstellten "C"-Quellcodes und Definitionsdateien können auf das ISaGRAF Zielsystem geladen werden, wenn es ein maschineneigenes Compiling-Tool unterstützt. Hierzu kann das Windows Standard **TERMINAL** Tool benutzt werden.

Wenn die Quelldateien auf dem Zielsystem verwaltet werden, müssen die Definitionsdateien mit einer neuen Ladeoperation aktualisiert werden, und zwar jedesmal, wenn eine Funktionsschnittstelle mit dem ISaGRAF Bibliotheksmanager modifiziert wird.

Die Befehlszeilen für das Laden der Dateien können beispielsweise in einer Batch-Datei zusammengefaßt werden und dann vom Workstation-Werkzeugmenü aus gestartet werden (siehe Benutzerhandbuch: Programmverwaltung).

Gebrauch eines Cross-Compilers

Die Quelldateien können auch direkt auf Ihrem PC verwaltet werden, wenn das Zielsystem ein PC ist oder wenn ein Cross-Compiler auf dem PC läuft und Code für das Zielsystem generiert.

In diesem Fall kann der Benutzer den ISaGRAF Bibliotheksmanager benutzen, um die Quellcodes der Umrechnungen, Funktionen und Funktionsbausteine zu vervollständigen und zu modifizieren.
Die Befehlszeilen für die Inbetriebnahme des Compilers und Linkers können beispielsweise in einer Batch-Datei zusammengefaßt werden und dann vom Workstation-Werkzeugmenü aus gestartet werden (siehe Benutzerhandbuch: Programmverwaltung).

Wenn die Umrechnungen, Funktionen und Funktionsbausteine auf dem PC kompiliert werden, wird der neu generierte (mit neuen Komponenten verbundene) ISaGRAF Kernel einfach vor der Ausführung der Anwendungen in das Zielsystem geladen. Wenn es sich bei dem Zielsystem um einen weiteren PC handelt, kann der neu generierte ISaGRAF Kernel mit Hilfe einer Diskette oder über ein Netzwerk in die Zielsteuerung geladen werden.

Verbindung mit den ISaGRAF Kernelbibliotheken

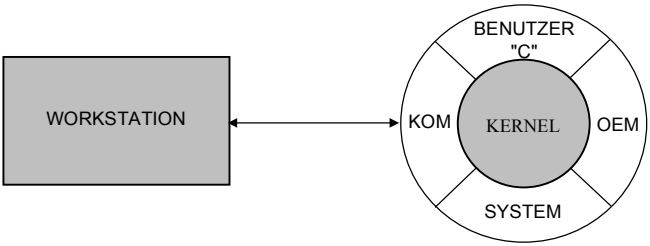
Warnung:
Es folgen allgemeine Informationen, die Ihrem Zielsystem möglicherweise nicht in allen Einzelheiten entsprechen.
Auf jeden Fall sollten Sie die Readme und.TXT Dateien Ihres Zielsystems einsehen.

Die ISaGRAF Zielsystem-Diskette enthält zahlreiche Utilities, mit denen Umrechnungen, Funktionen und Funktionsbausteine kompiliert und mit den Bibliotheken des ISaGRAF Kernels verbunden werden können.

Es gibt zwei Implementierungen:
- Einzeltask-ISaGRAF: alle Funktionen werden in einem einzigen Programm ausgeführt
- Multitask-ISaGRAF: ein separater Task (Thread) ist der Kommunikation dediziert

In beiden Fällen sind die "C"-Komponenten in den gleichen Bibliotheken untergebracht. Für den "C"-Programmierer besteht zwischen Einzeltasking und Multitasking also kein Unterschied. In einer Einzeltaskversion sind die "C"-Benutzerbibliotheken mit einem einzelnen Task verbunden (gewöhnlich **isa** genannt), im Gegensatz zu der Multitaskversion, in der die Bibliotheken mit dem Kernel-Task (gewöhnlich **isaker** genannt) verbunden sind.

Entwicklungs-system	**Ziel-system**

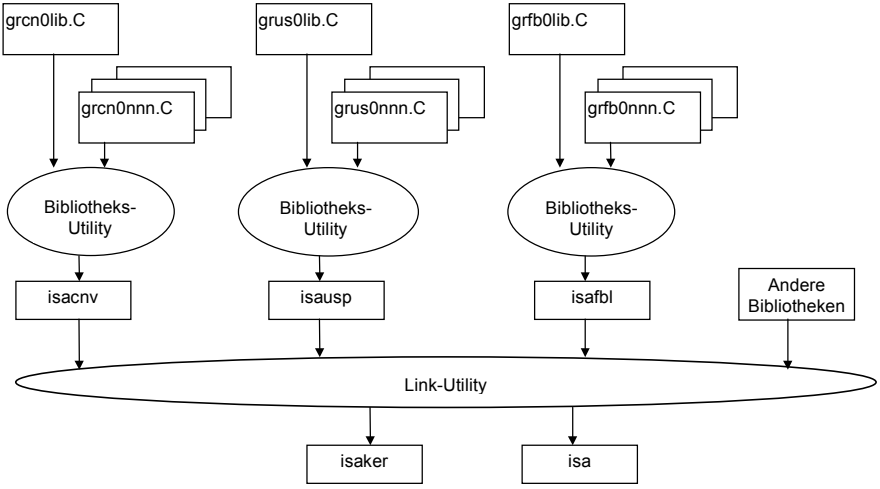


Der Kernel der ISaGRAF Software ist hardware-unabhängig. Er führt die IEC-Sprachen aus und besitzt seine eigene Variablen-Datenbank.

Der erste Schritt bei der Verbindung mit dem Kernel ist die Erstellung von Bibliotheken für sämtliche Umrechnungen, Funktionen und Funktionsbausteine des Projekts:

Bibliothek	Inhalt
ISAUSP	- GRUS0LIB Objektdatei (Feld deklarierter Funktionen) - Objektdatei für jede integrierte Funktion
ISAFBL	- GRFB0LIB Objektdatei (Feld deklarierter Funktionsbausteine) - Objektdatei für jeden integrierten Funktionsbaustein
ISACNV	- GRCN0LIB Objektdatei (Feld deklarierter Umrechnungen) - Objektdatei für jede integrierte Umrechnung

Daraufhin muß der Programmierer diese neuen Bibliotheken mit den anderen Objektdateien und Bibliotheken des ISaGRAF Kernels verbinden. Die einzelnen Schritte für die Integration der "C"-Komponenten werden im folgenden Diagramm veranschaulicht:



Es folgt eine Liste sämtlicher Objektmodule und Bibliotheken, die bei der Verbindung berücksichtigt werden müssen:

Für isaker:		
Objektmodul:	**tast0mai**	
Objektmodul:	**tats0com**	
Kernel-Bibliothek:	**isaker**	
Kernel-Bibliothek:	**isaoem**	
Benutzer-Bibliothek:	**isausp**	benutzerdefinierte Funktionen
Benutzer-Bibliothek:	**isafbl**	benutzerdefinierte F-Bausteine
Benutzer-Bibliothek:	**isacnv**	benutzerdefinierte Umrechnungen
Kernel-Bibliothek:	**isasys**	
Systembibliotheken:	(siehe "C"-Compiler-Handbuch)	

Für isa:		
Objektmodul:	**tast0mai**	
Objektmodul:	**tast0com**	
Kernel-Bibliothek:	**isaker**	
Kernel-Bibliothek:	**isatst**	
Kernel-Bibliothek:	**isaoem**	
Benutzer-Bibliothek:	**isausp**	benutzerdefinierte Funktionen
Benutzer-Bibliothek:	**isafbl**	benutzerdefinierte F-Bausteine
Benutzer-Bibliothek:	**isacnv**	benutzerdefinierte Umrechnungen
Kernel-Bibliothek:	**isasys**	
Systembibliotheken:	(siehe "C"-Compiler-Handbuch)	

Der Programmierer muß möglicherweise die genaue Reihenfolge der oben aufgelisteten Objektmodule und Bibliotheken einhalten. Entsprechend des jeweiligen Zielsystems besitzen die Objektmodule und Bibliotheken Standard-Erweiterungen ("**.lib**", "**.obj**", "**.l**", "**.r**"...).

Compiling- und Link-Optionen

Bei der Kompilierung und Verbindung können bestimmte Optionen ausgewählt werden, die vom Typ der von den Umrechnungen, Funktionen und Funktionsbausteinen verarbeiteten Operationen abhängen. Manche Operationen erfordern andere (mathematische, grafische...) Systembibliotheken für die Verbindung.

Sämtliche "C"-Quelldateien des ISaGRAF Kernels wurden mit dem Speichermodell **LARGE** kompiliert. Dieses Modell muß auch für die Kompilierung der Umrechnungen, Funktionen und Funktionsbausteine benutzt werden.

Eine spezielle Konstante muß für die Kompilierung der "C"-Bibliothekskomponenten definiert werden. Sie kennzeichnet den Typ des Zielsystems und Prozessors, so daß der Quellcode

der Umrechnungen, Funktionen und Funktionsbausteine system-unabhängig bleibt. Es folgen die Namen dieser konstanten Werte:

DOS.....für Systeme auf DOS-Basis (INTEL Prozessor)
ISAWNT .....für Systeme auf Windows-NT-Basis (INTEL Prozessor)
OS9 .....für OS9-Systeme (MOTOROLA Prozessor)
VxWorks .....für VxWorks-Systeme (MOTOROLA Prozessor)

Die mit der ISaGRAF Zielsystem-Software gelieferten Utility-Befehlsdateien (für die Kompilierung und Verbindung) zeigen, wie die entsprechende Konstante in der Compiler-Befehlszeile definiert wird.

Unterstützte Compiler

Die folgenden Compiler werden für die Entwicklung von Umrechnungen, Funktionen und Funktionsbausteine und die Verbindung mit dem ISaGRAF Kernel unterstützt:

Microsoft MSC 7.00 Compiler	für Zielsysteme auf DOS-Basis
Microsoft MSVC 4.00 Compiler	für Zielsysteme auf Windows-NT-Basis
Microware ULTRA-C Compiler	für OS-9 Zielsysteme
Tornado 1.0; GNU Toolkit 2.6	für VxWorks Zielsysteme

Wenden Sie sich an CJ International, wenn Sie andere Compiler benutzen möchten.

Zusammenfassung

Es folgt die Zusammenfassung der Operationen, die für die Entwicklung neuer Umrechnungen, Funktionen und Funktionsbausteine erforderlich sind.

⇨1. Erstellen Sie ein neues Element mit dem ISaGRAF Bibliotheksmanager, benennen Sie es und versehen Sie es mit einen Kommentar. Der Block der "C"-Quelldatei wird automatisch generiert.

⇨2. Beschreiben Sie die Schnittstelle (Aufruf- und Returnparameter) mit dem ISaGRAF Bibliotheksmanager, wenn es sich um eine Funktion oder einen Funktionsbaustein handelt. Der "C"-Quellcodevorsatz wird automatisch generiert.

⇨3. Erfassen Sie des Datenblatt (detaillierte Benutzeranleitung) für das Element.

⇨4. Vervollständigen Sie die "C"-Quelldatei mit dem ISaGRAF Bibliotheksmanager, indem Sie die "C"-Programmierung des Algorithmus der Umrechnung, der Funktion oder des Funktionsbausteins eingeben. Der Quellcode des Elements ist jetzt abgeschlossen. (Hierzu kann auch ein anderer Editor benuzt werden.)

⇨5. Im Bibliotheksmanager wählen Sie die Option "**Logische Nummer anzeigen**", um festzustellen, welche logische Nummer dem neuen Element zugeordnet wurde. Diese Nummer wird in den Pfadnamen der entsprechenden ".C" und ".H" Quelldateien benutzt.

⇨6. Kopieren/laden Sie die .C und .H Dateien in Ihr Zielsystem (bei maschineneigenem Compiler) oder in das entsprechende Umfeld (bei Cross-Compiler), in dem die ISaGRAF Zielsystembibliotheken und -Tasks installiert sind.

⇨7. Führen Sie den "C"-Compiler auf der neuen Quelldatei aus und berichtigen Sie eventuelle Syntaxfehler.

⇨8. Fügen Sie den Namen des Deklarationsservice des neuen Elements in die Quelldatei "**GR??0LIB.C**" ein, die das Feld eingefügter Elemente definiert.

⇨9. Führen Sie den "C"-Compiler aus, um die Datei "**GR??0LIB.C**" zu kompilieren.

⇨10. Fügen Sie den Namen des Objektmoduls in die Liste der Objektdateien ein, die für den Aufbau der entsprechenden Bibliothek benutzt werden.

⇨11. Starten Sie die "C"-Bibliothekserstellung. Starten Sie den "C"-Linker, um den neuen Kernel aufzubauen.

⇨12. Installieren Sie den neu erstellten Kernel in Ihrer Zielsteuerung.

⇨13. Schreiben Sie eine ISaGRAF Beispeilanwendung, um die Aktivierung und die Schnittstelle des neuen Elements zu testen.

C.8. Modbus-Verbindung

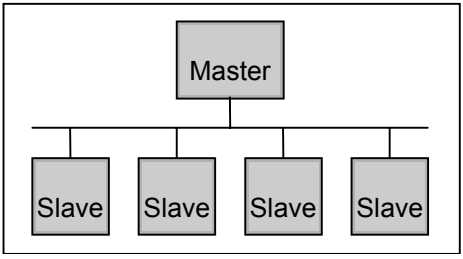
Wenn die Anwendung vollständig entwickelt und getestet ist, kann sie an ein Prozeßvisualisierungssystem angeschlossen werden.

ISaGRAF ist ein offenes System mit vielseitigen Netzwerkbetriebsmöglichkeiten.
Das einfachste industrielle Netzwerk ist das MODBUS/MODICON Standardprotokoll, das auf fast allen Prozeßvisualisierungssystemen vorhanden ist und das nur eine serielle Verbindung benötigt (RS232, RS485, Current Loop).

Das ISaGRAF Debuggerkommunikationsprotokoll ist MODBUS-kompatibel und ermöglicht Lese- und Schreibzugriff von einem Modbus-Master auf die Variablen.

C.8.1. MODBUS-Netzwerk und -Protokoll

Ein Modbus-Netzwerk besteht aus nur einer Master-Station (gewöhnlich ein Prozeßvisualisierungssystem) und einer oder mehrerer Slave-Stations (gewöhnlich SPS).



Der Master sendet eine Anfrage zur Zeit an einen Slave (unter Anwendung seiner Slave-Nummer) und wartet auf die Antwort des Slaves, bevor er die nächste Anfrage sendet. Die nicht-betroffenen Slaves antworten nicht.

Jeder Block enthält eine Slave-Nummer, eine Anfrage-Nummer und entsprechende Daten und ein 16 Bit-Checksum (CRC).

Wenn keine Antwort nach einem Time-Out erhalten wird, kann die Anfrage mehrere Male wiederholt werden, bevor der Master die Verbindung zum Slave als "abgebrochen" erklärt.
Der Time-Out-Wert und die Anzahl der Anfrageversuche wird auf der Master-Station eingestellt, um den Anforderungen der Slaves zu entsprechen (abhängig von der Anwendung, usw...).

Wenn bei der Anfrageverarbeitung ein Fehler auftritt, sendet der Slave unter Umständen eine Fehlermeldung anstatt des erwarteten Antwortblocks.

Modbus ist ein Modicon-Protokoll, jedoch keine internationale Norm.

Es gibt viele unterschiedliche Implementierungen von 'Modbus'-kompatiblen Protokollen mit vielen Möglichkeiten, zum Beispiel:

– Liste implementierter Funktionscodes
– Adressenabbildung (Mapping)
– RTU (binäre Codes) oder ASCII-Protokoll
usw...

C.8.2. ISaGRAF Implementierung

Zugriff auf Anwendungsvariablen

Die ISaGRAF Kommunikationsverbindung erkennt fünf Modbus-Funktionscodes:

1	N Bits lesen
3	N Wörter lesen
5	1 Bit schreiben
6	1 Wort schreiben
16	N Wörter schreiben

Es besteht die Möglichkeit, auf die ISaGRAF Anwendungsvariablen über ihre "Netzwerkadressen" zuzugreifen, wenn sie im Datenverzeichnis der Workstation definiert worden sind. Dies ist bei den folgenden Variablen möglich:
- Boolesche oder analoge Variablen
- Eingangs-, Ausgangs- oder interne Variablen
- lokale oder globale Variablen.

Um eine boolesche Variable zu schreiben, werden die Funktionen 5, 6 oder 16 benutzt. Ein TRUE-Wert für das Schreiben ist ein beliebiger Wert ungleich null.

Um eine boolesche Variable zu lesen, können die Funktionen 1 oder 3 benutzt werden. Wenn die Funktion 1 benutzt wird, werden die Werte als Bit-Feld abgerufen. Wenn die Funktion 3 benutzt wird, werden die Werte in Bytes abgerufen (ein TRUE-Wert entspricht 0xFFFF).
Um eine analoge Variable zu schreiben, werden die Funktionen 6 oder 16 benutzt. Der Wert ist eine 16 Bit Ganzzahl zwischen -32768 und +32767 (ISaGRAF Zielsystemvariablen besitzen 32 Bits).

Um eine analoge Variable zu lesen, sollte die Funktion 3 benutzt werden. Der gelesene Wert ist eine 16 Bits Ganzzahl zwischen -32768 und +32767. Auf dem Zielsystem besitzen die analogen Variablen 32 Bits. Daher wird ein Wert, der 16 Bits überschreitet (positiv oder negativ), auf dem Zielsystem mit dem maximalen 16 Bit-Wert gelesen (positiv oder negativ).

Es ist nicht möglich, mit einer Modbus-Anfrage auf reale Variablen zuzugreifen.

Warnung:
Die ISaGRAF Implementiering verwaltet keine Fehlercodes vom Typ "unbekannte Modbus-Adresse".

Notationen:

slv	slave number (Slave-Nummer)
nbw	number of words (Anzahl der Wörter)
nbb	number of bytes (Anzahl der Bytes)
nbi	number of bits (Anzahl der Bits)
addH	network address - High byte (Netzwerkadresse - hochwertiges Byte)
addL	network address - Low Byte (Netzwerkadresse - niedrigwertiges Byte)
vH	value - High Byte (Wert - hochwertiges Byte)
vL	value - Low Byte (Wert - niedrigwertiges Byte)
V	Byte Value (Byte-Wert)
bfd	Bit field - nbb Bytes (Bitfeld - nbb Bytes)
crcH	checksum - High Byte (Checksum - hochwertiges Byte)
crcL	checksum - Low byte (Checksum - niedrigwertiges Byte)

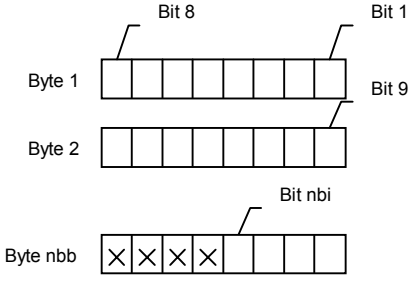
FUNKTION 1: N Bits lesen

nbi Bits lesen (boolesch), angefangen bei Netzwerkadresse addH/addL

Anfrage	slv	01	addH	addL	00	nbi	crcH	crcL

Antwort	slv	01	nbb	bfd	...		crcH	crcL
				Byte 1		Byte nbb		

bfd ist ein Bitfeld von nbb Bytes mit dem folgenden Format:



Bit 1 entspricht dem Wert der Variablen bei Netzwerkadresse addH/addL.
Bit nbi entspricht dem Wert der Variablen bei Netzwerkadresse addH/addL + nbi -1.
X bedeutet undefinierter Wert.

FUNKTION 3: N Wörter lesen

nbw Wörter lesen, angefangen bei Netzwerkadresse addH/addL

Anfrage	slv	03	addH	addL	00	nbw	crcH	crcL

Antwort	slv	03	nbb	vH	vL	...	crcH	crcL

nbb entspricht der Anzahl der Bytes vH, vL.

FUNKTION 5: 1 Bit schreiben

Ein Bit schreiben (boolesch) an Netzwerkadresse addH/addL

Anfrage	slv	05	addH	addL	vH	00	crcH	crcL

Antwort	slv	05	addH	addL	vH	00	crcH	crcL

FUNKTION 6: 1 Wort schreiben

Ein Wort schreiben an Netzwerkadresse addH/addL

Anfrage	slv	06	addH	addL	vH	vL	crcH	crcL

Antwort	slv	06	addH	addL	vH	vL	crcH	crcL

FUNKTION 16: N Wörter schreiben

nbw Wörter schreiben, angefangen an Netzwerkadresse addH/addL (nbb = 2nbw)

Anfrage	slv	10	addH	addL	00	nbw	nbb	vH	vL	...	crcH	crcL

Antwort	slv	10	addH	addL	00	nbw	crcH	crcL

Beispiele:

- Funktion 1: 15 bits lesen, angefangen an Netzwerkadresse 0x1020, auf Slave 1

Anfrage	01	01	10	20	00	0F	79	04

Antwort	01	01	02	00	12	39	F1

Der gelesene Wert ist 0x0012, was 00000000 00010010 als Bitfeld ergibt.
In diesem Beispiel sind die Variablen 0x1029 und 0x102C TRUE, alle anderen sind FALSE.

- Funktion 16: 2 Wörter schreiben an Adresse 0x2100 auf Slave 1. Die geschriebenen Werte sind 0x1234 und 0x5678.

Anfrage	01	10	21	00	00	02	04	12	34	56	78	1C	CA

Antwort	01	10	21	00	00	02	4B	F4

Dateiübertragung

Im Vergleich zu modernen Feldbussen leistet das Modbus-Protokoll ausgesprochen schlechte Dienste, wenn es nicht mit speziellen Hersteller-Funktionscodes ergänzt wird.

In unserer Situation, wenn ISaGRAF auf einer leistungsfähigen und flexiblen Computerbasis betrieben wird, gibt es zwei Einschränkungen zum Modbus-Protokoll:
- Es kann nur auf ISaGRAF Variablen zugegriffen werden
- Es ist schwierig, eine schnelle Übertragung mit einer großen Anzahl an Daten auszuführen

Aus diesem Grunde verfügt ISaGRAF über einen Satz von "Modbus-ähnlichen" Anfragen zur Dateiübertragung und ein "Remote-Dateiverwaltungs"-Protokoll. Diese Funktionen müssen für die folgenden Operationen implementiert werden:
- Laden einer binären oder ASCII-Datei
- Rücklesen einer binären oder ASCII-Datei
- Dynamischer Datenaustausch über eine virtuelle oder physische, gemeinsam benutzte Datei.

Auf diese Weise kann eine beliebige "ISaGRAF unabhängige" Anwendung mit Hilfe der ISaGRAF Kommunikationsverbindung problemlos mit einem Remote-Zielsystem kommunizieren.

Das Protokoll stützt sich auf die folgenden Konzepte:
- Die Datei auf dem ISaGRAF Zielsystem heißt **remote file**
- Die Datei auf dem Master-Computer heißt **local file**
- Auf jedes Byte einer Datei wird mit einer 32 Bit **Basisadresse** und einer 16 Bit **Byteadresse** zugegriffen.

Es gibt Anfragen für die Auswahl des Remote-Dateinamens und der Basisadresse und zum Lesen oder Schreiben der Daten der Remote-Datei unter Anwendung der 16 Bit Byte-Adresse.

FUNKTION 17: Daten schreiben

nbb entspricht der Anzahl an vH, vL Bytes

Anfrage	slv	11	addH	addL	00	nbb	nbb	vH	vL	...	crcH	crcL

Antwort	slv	11	addH	addL	00	nbb	crcH	crcL

Die Bedeutung dieser Anfrage ist unterschiedlich, je nach Adressenbereich addH/addL:

- **0xF000: Remote-Dateinamen initialisieren**
  nbb entspricht der Anzahl der Zeichen für den Dateinamen, spezifiziert in den Feldern vH vL (in diesem Fall haben High und Low keine Bedeutung) und **einschließlich \0** für das Ende der Zeichenkette.
  Wenn die Datei nicht existiert, wird sie mit den Attributen "writable + readable + executable" (schreibbar, lesbar, ausführbar) erstellt.

- **0xF002: Ändern der Basisadresse auf einen spezifizierten Wert**
  nbb sollte gleich 4 sein. Das erste vH/vL Byte entspricht dem hochwertigen Wort des spezifizierten Werts. Beliebige 32 Bit Werte werden akzeptiert.
  Alle zukünftigen Lese- und Schreibanfragen benutzen diese Basisadresse. Wenn diese Anfrage nicht benutzt wird, ist der vorgegebene Wert der Basisadresse null.

- **0xF004: Datei löschen**
  nbb sollte gleich null sein.
  Die Datei wird gelöscht, wenn sie existiert und wenn dies möglich ist.

- **Größer als 0xF004: Reserviert**

- **Kleiner als 0xF000: Bytes schreiben**
  Die spezifizierte Adresse, an der die Bytes geschrieben werden sollen, wird in addH/addL spezifiziert. Sie muß kleiner als F000 sein. Die in vH vL Feldern spezifizierten Bytes (nbb Bytes), in denen High und Low möglicherweise keine Bedeutung mehr haben, werden in der gegebenen Anordnung (von links nach rechts) an den zuvor ausgewählten Remote-Dateinamen geschrieben. Die Startadresse, an die geschrieben wird, ist die spezifizierte Adresse, die der zuvor ausgewählten Basisadresse hinzugefügt wurde. Wenn der sich ergebende Adressenzugriff den aktuellen Umfang der Datei überschreitet, wird die Datei erweitert. Der Dateiumfang kann nicht verringert werden.

FUNKTION 18: Daten lesen

Anfrage	slv	12	addH	addL	00	nbb	crcH	crcL

Antwort	slv	12	nbb	V	V	...	crcH	crcL

Die spezifizierte Adresse, an der Bytes gelesen werden sollen, wird in addH/addL spezifiziert. Sie muß kleiner als F000 sein. Die spezifizierte Anzahl an Bytes (nbb) wird vom zuvor ausgewählten Remote-Dateinamen gelesen, angefangen an der spezifizierten Adresse (addH/addL mit beliebigem 16 Bit-Wert), die der zuvor ausgewählten Basisadresse hinzugefügt wurde.
Die Werte werden in der Reihenfolge abgerufen (V Felder von links nach rechts), in der sie in der Datei gelesen werden.

Beispiel:

Auswahl des Remote-Dateinamens: 'target.fil'.

Anfrage	01	11	F0	00	00	0B	0B	74	...	00	25	9F

Antwort	01	11	F0	00	00	0B	8F	0E

Auswahl der Basisadresse: 0x10000.

Anfrage	01	11	F0	02	00	04	04	00	01	00	00	76	11

Antwort	01	11	F0	02	00	04	6E	CA

4 Bytes schreiben: absolute Adresse 0x107D0, Werte 01,02,03,04.

Anfrage	01	11	07	D0	00	04	04	01	02	03	04	28	6F

Antwort	01	11	07	D0	00	04	FC	87

4 Bytes lesen: absolute Adresse 0x107D0.

Anfrage	01	12	07	D0	00	04	B8	87

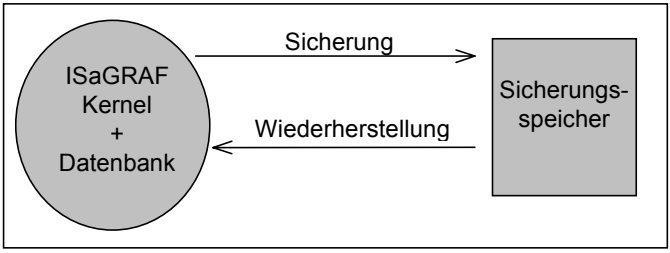
Antwort	01	12	04	01	02	03	04	58	7D

C.9. Verwaltung von Stromausfällen

C.9.1. Grundlagen

Stromausfälle sind äußerst kritisch im Leben einer Anwendung. Die Verwaltung eines Stromausfalls wird durch die folgenden Faktoren bedingt:
– Prozeßspezifizierungen
– Hardware-Kapazitäten
– Programmierungsmethoden.

Aus diesem Grunde ist die ISaGRAF Lösung für die Verwaltung von Stromausfällen keine Universalmethode, sondern eine Gruppe von Prinzipien, Methoden und Werkzeugen, die speziell für jede Anwendung, oder zumindest die Hardware, miteinander kombiniert werden müssen.
Um einem Prozeßsteuerungssystem die Möglichkeit zu geben, nach einem Stromausfall wieder richtig zu starten, müssen 3 Probleme gelöst werden:



– Datensicherung
– Erkennung eines vorgefallenen Stromausfalls beim Start
– Wiederherstellung der gesicherten Daten

Für das zweite Problem gibt es keine Standard-Software-Lösung. Der Systemlieferant könnte möglicherweise Werkzeuge zur Verfügung stellen, die den Zugriff von einer ISaGRAF Anwendung oder einem C-Programm auf den Hardware-Status ermöglichen.

Es ist wichtig, zu entscheiden, welche Daten gesichert und wiederhergestellt werden sollen. Hierbei unterscheidet man zwischen zwei Arten von Daten:
– Anwendungsvariablen:
  - Prozeßvariablen, wie beispielsweise Anzahl der verarbeiteten Elemente, Datum des Stromausfalls, Werte der Anwendungsparameter, usw....
  - Programmvariablen, wie Zähler, Timer, Zwischenwerte und Flags.
– Programmstatus:
  zum Beispiel die Referenz der aktiven Schritte, der Status eines jeden C-Programms...

Diese beiden Fälle und die jeweiligen ISaGRAF Lösungen werden in den folgenden Kapiteln behandelt.

C.9.2. Sicherung der Anwendungsvariablen

Nicht-flüchtige Variablen

Mit dem Workstation-Variableneditor hat man die Möglichkeit, für jede interne Variable (nicht E/A) das Attribut "nicht-flüchtig" auszuwählen.

Am Ende eines jeden Zielsystemzyklus werden die Werte der nicht-flüchtigen Variablen in einen speziellen Speicher kopiert, gewöhnlich ein batterie-unterstütztes RAM.

Wenn beim Start mindestens eine Variable das Attribut "nicht-flüchtig" hat, sucht ISaGRAF nach nicht-flüchtigen Variablen:

- Wenn die Anwendung zuvor gelaufen ist, erkennt ISaGRAF die gespeicherten Werte und ordnet sie allen nicht-flüchtigen Variablen zu.
- Wenn die vorherige Anwendung eine andere war oder wenn zuvor keine Anwendung ausgeführt wurde, erkennt ISaGRAF, daß die "nicht-flüchtigen" Werte ungültig sind und setzt sämtliche "nicht-flüchtigen" Variablen auf null zurück.

Die Speicherzuordnung für die Sicherung der verschiedenen Variablentypen erfolgt in der Workstation, in den **Ausführungsparametern** unter **nicht-flüchtige Variablen.**
Die spezifizierte Zeichenkette muß das folgende Format haben:

boo_add , boo_size , ana_add , ana_size , tmr_add , tmr_size , msg_add , msg_size

mit:

boo_add: Hexadezimale Adresse, die benutzt wird, um boolesche Variablen zu speichern. Muß immer ungleich null sein.

boo_size: Hexadezimale Größe in Bytes, die an dieser Adresse verfügbar ist. Es wird ein Byte zum Sichern von jeweils einer booleschen Variablen benötigt.

ana_add: Hexadezimale Adresse, die benutzt wird, um analoge Variablen zu speichern. Muß immer ungleich null sein.

ana_size: Hexadezimale Größe in Bytes, die an dieser Adresse verfügbar ist. Es werden immer mindestens 4 Bytes benötigt, mit zusätzlich 4 Bytes für jede zu speichernde analoge Variable.

tmr_add: Hexadezimale Adresse, die benutzt wird, um Timer-Variablen zu speichern. Muß immer ungleich null sein.

tmr_size: Hexadezimale Größe in Bytes, die an dieser Adresse verfügbar ist. Es werden 5 Bytes benötigt, um eine Timer-Variable zu speichern.

msg_add: Hexadezimale Adresse zum Speichern vom Zeichenketten-Variablen. Muß immer ungleich null sein.

msg_size: Hexadezimale Größe in Bytes, die an dieser Adresse verfügbar ist. Es werden 256 Bytes benötigt, um eine Zeichenketten-Variable zu speichern.

Anforderungen:

– Alle Felder aller Typen müssen spezifiziert werden, sogar wenn nicht alle Variablentypen gesichert werden müssen. In diesem Fall muß eine Größe gleich null (außer für analoge Variablen, für die eine Größe von 4 Bytes spezifiziert werden muß) und eine beliebige Adresse ungleich null für den (die) nicht erforderlichen Variablentyp(en) spezifiziert werden.

Beispiel:

Nehmen wir einmal an, daß die folgenden Variablen gesichert werden sollen:

20 Boolesche Variablen
0 Analoge Variablen
0 Timer-Variablen
3 Zeichenketten-Variablen

Der Sicherungsspeicher befindet sich an der Hexadezimaladresse 0xA2F200.

Nehmen wir weiter an, daß:

Die booleschen Variablen an Adresse 0xA2F200 mit der genau erforderlichen Größe von 20 Bytes gespeichert werden.

Die Mindestgröße von 4 Bytes für die analogen Variablen an der Adresse 0xA2F214 gespeichert wird.

Die Timer-Dummy-Adresse 0xA2F200 ist und mit einer Größe gleich null spezifiziert wird.

Die Zeichenketten an der Adresse 0x A2F218 mit der genau erforderlichen Größe von 256*3 Bytes gespeichert werden.

Dann müßte die in die Workstation eingegebene Zeichenkette so aussehen:

```
A2F200,14,A2F214,4,A2F200,0,A2F218,300
```

SYSTEM-Funktionsaufruf

Wenn die meisten der Anwendungsvariablen gespeichert werden sollen, dann sollte die SYSTEM-Funktion benutzt werden, um mit einem kompletten Satz an Variablen umzugehen (weitere Informationen über die SYSTEM-Funktion finden Sie im Benutzerhandbuch). Beachten Sie, daß in einem solchen Fall sowohl die Sicherung als auch die Wiederherstellung vom Programmierer auf Anwendungsebene verwaltet wird.

Zuerst muß der Adressenbereich für die Sicherung des spezifizierten Variablentyps oder aller Variablentypen definiert werden:

<neue_adresse> := SYSTEM(SYS_INITxxx,<adresse>);

wobei:

– <adresse> der Sicherungs-Adressenbereich ist (16# Wert für Hexadezimalformat). Die Adresse muß gerade sein, sonst mislingt die Operation.
– SYS_INITxxx kann sein:
  * SYS_INITBOO für die Definition des Adressenbereichs aller booleschen Variablen
  * SYS_INITANA für die Definition des Adressenbereichs aller analogen Variablen.
  * SYS_INITTMR für die Definition des Adressenbereichs aller Timer-Variablen.
  * SYS_INITALL für die Definition des Adressenbereichs aller booleschen, analogen und Timer-Variablen.
– <neue_adresse> findet die nächste freie Adresse, d.h. <adresse> + Größe der gespeicherten Variablen (in Bytes) gemäß SYS_INITxxx. Auf diese Weise kann die erforderliche Sicherungsadressengröße geprüft werden. Wenn die Operation mislingt, ergibt <neue_adresse> null.

Dann kann eine Sicherung beantragt werden. Diese Prozedur kann jederzeit in der Anwendung aufgerufen werden. Die Sicherung findet am Ende des aktuellen Zyklus und nur einmal statt. Wenn die Hardware über einen booleschen Eingang oder eine C-Funktion verfügt, die den Benutzer informiert, wenn ein Stromausfall eintritt, und mindestens eine ISaGRAF Zyklusverzögerung vor dem Zusammenbruch gewährt wird, dann kann die Sicherung erst stattfinden, wenn ein Stromausfall erkannt wurde:

<Fehler> :=SYSTEM(SYS_SAVxxx,0);

wobei:
- SYS_SAVxxx sein kann:
  - SYS_SAVBOO, um die Sicherung aller booleschen Variablen zu beantragen.
  - SYS_SAVANA, um die Sicherung aller analogen Variablen zu beantragen.
  - SYS_SAVTMR, um die Sicherung aller Timer-Variablen zu beantragen.
  - SYS_SAVALL , um die Sicherung aller booleschen, analogen und Timer-Variablen zu beantragen.
- <Fehler> nimmt einen Fehlerstatus ungleich null an, wenn die Operation mislungen ist (SYS_INITxxx wurde nicht aufgerufen).

Schließlich müssen die Variable wiederhergestellt werden. Diese Prozedur kann jederzeit in der Anwendung aufgerufen werden. Die Wiederherstellung findet am Ende des aktuellen Zyklus und nur einmal statt. Um sicher zu gehen, daß die gesicherten Daten gültig sind, sollte eine analoge Variable auf einen konstanten Wert gesetzt und als Signatur benutzt werden:

<Fehler> := SYSTEM(SYS_RESTxxx,0);

wobei:
- SYS_RESTxxx sein kann:
  - SYS_RESTBOO für die Wiederherstellung aller booleschen Variablen.
  - SYS_RESTANA für die Wiederherstellung aller analogen Variablen.
  - SYS_RESTTMR für die Wiederherstellung aller Timer-Variablen.
  - SYS_RESTALL für die Wiederherstellung aller booleschen, analogen und Timer-Variablen.
- <Fehler> nimmt einen Fehlerstatus ungleich null an, wenn die Operation mislungen ist (SYS_INITxxx wurde nicht ausgeführt).

Es folgt die Zusammenfassung der Befehle der SYSTEM-Funktion für die Verwaltung der Variablensicherung:

Befehl		**Bedeutung**
vordefiniertes Schlüsselwort	**Wert**	
SYS_INITBOO	16#20	init Sicherung boolescher Variablen
SYS_SAVBOO	16#21	boolesche Variablen sichern
SYS_RESTBOO	16#22	boolesche Variablen wiederherstellen
SYS_INITANA	16#24	init Sicherung analoger Variablen
SYS_SAVANA	16#25	analoge Variablen sichern
SYS_RESTANA	16#26	analoge Variablen wiederherstellen
SYS_INITTMR	16#28	init Sicherung Timer-Variablen
SYS_SAVTMR	16#29	Timer-Variablen sichern
SYS_RESTTMR	16#2A	Timer-Variablen wiederherstellen
SYS_INITALL	16#2C	init Sicherung aller Typen
SYS_SAVALL	16#2D	alle Typen sichern

SYS_RESTALL	16#2E	alle Typen wiederherstellen

Befehl (vordefiniertes Schlüsselwort)	**Argument**	**Rückgabewert**
SYS_INITxxx	Speicheradresse	nächste freie Adresse
SYS_SAVxxx	0	null, wenn OK
SYS_RESTxxx	0	null, wenn OK

Kundenspezifische Implementierung

Mit Hilfe von C-Funktionen und -Funktionsbausteinen können spezifische Benutzer-Prozeduren entwickelt werden, die auf einen batteriebetriebenen Sicherungsspeicher zugreifen, um Variablen zu einem beliebigen Zeitpunkt in der Anwendungsausführung zu speichern und wiederherzustellen.

Beispiele:

1) Anwendungsspezifische Prozedur:
backup, restore_temp, restore_date, restore_cnt sind benutzerdefinierte C-Prozeduren.

backup(temperature, date, cnt); 3 kritische Daten speichern

temperature := **restore_temp()**; Temperatur wiederherstellen
date := **restore_date()**; Datum wiederherstellen
cnt := **restore_cnt()**; Zähler wiederherstellen

2) Allgemeine Prozeduren:
backup_init, backup, backup_link, restore sind benutzerdefinierte C-Prozeduren.

save_id := **backup_init**(address, size); speichergesichertes Feld zuweisen.
backup(save_id, cpt1, 3); cpt1 als das 3. Element sichern.

rest_id := **backup_link**(address, size) gesicherten Speicher verbinden.
cpt1 := **restore**(rest_id, 3); gesicherten Wert von cpt1 wiederherstellen.

C.9.3. Sicherung des Programmstatus

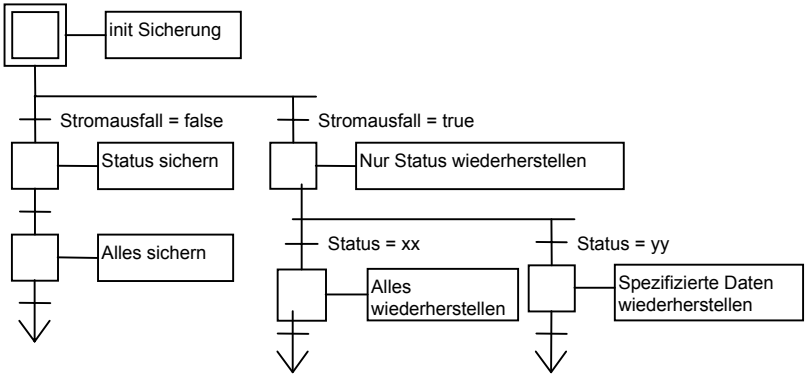
Es ist durchaus möglich, jeden Status eines jeden Anwendungsprogramms zu speichern, aber es kann gefährlich sein, jedes Programm mit dem Status der letzten Sicherung wiederherzustellen, aus mindestens drei Gründen:

- Manche Prozesse erfordern spezifische Operationen vor einem Neustart.
- Es ist langwierig, sich mit jedem Status einer vollständigen Anwendung zu befassen.
- Manche externen Ressourcen, z.B. C-Programme, Peripherien, usw. können nicht automatisch wieder gestartet werden.

Die beste Lösung für die Beschreibung des Prozeßstatus scheint die Sicherung der analogen und booleschen Variablen zu sein, wenn der Programmierer der Meinung ist, daß diese Informationen in der Wiederanlaufphase brauchbar sind.
Durch ein unvollständiges und doch intelligentes "Bild" des Prozesses sollte es dann möglich sein, AS-Programme zu starten, zu stoppen oder anzuhalten und Variablen zu initialisieren,

um die Anwendung auf einen angemessenen Status zu bringen. Aber ISaGRAF liefert keine automatische Anlaufprozedur.

Beispiel:



C.10. Anhang: Fehlerliste und Beschreibung

Fehlerliste:

Code	Meldung	Typ
1	nicht genug Speicherplatz, um die Datenbank zu speichern	System
2	unzusammenhängende Datenbank (Motorola/Intel)	Anwendung
3	nicht genug Speicher, um die Kommunikations-Mailbox zu installieren	System
4	Verknüpfung mit der Datenbank unmöglich	System
5	Time-Out während einer Anfrage an den Kernel	System
6	Time-Out während einer Antwort vom Kernel	System
7	Installation der Kommunikation unmöglich	System
8	kann nicht-flüchtigen Variablen keinen Speicher zuteilen	Anwendung
9	Anwendung gestoppt	Anwendung
10	zu viele N- oder P-Aktionen gleichzeitig	Anwendung
11	zu viele Setze-Aktionen gleichzeitig	Anwendung
12	zu viele Rücksetze-Aktionen gleichzeitig	Anwendung
13	TIC-Anweisung unbekannt	Anwendung
16	Antwort auf eine Leseanforderung unmöglich	System
17	Antwort auf eine Schreibanforderung unmöglich	System
18	Antwort auf Session-Anforderung unmöglich	System
19	Antwort auf Modbus-Anforderung unmöglich	System
20	Antwort an Debugger unmöglich	System
21	Antwort an Debugger unmöglich	System
23	Anforderungs-Code unbekannt	System
24	Ethernet-Kommunikationsfehler	System
25	Fehler der Kommunikationssynchronisierung	System
28	nicht genug Speicher für die Anwendung	System
29	nicht genug Speicher für die Anwendungsaktualisierung	System
30	OEM-Schlüsselcode unbekannt	Anwendung
31	Initialisierung einer booleschen Eingangskarte unmöglich	Anwendung
32	Initialisierung einer analogen Eingangskarte unmöglich	Anwendung
33	Initialisierung einer Eingangskarte vom Typ Zeichenkette unmöglich	Anwendung
34	Initialisierung einer booleschen Ausgangskarte unmöglich	Anwendung
35	Initialisierung einer analogen Ausgangskarte unmöglich	Anwendung
36	Initialisierung einer Ausgangskarte vom Typ Zeichenkette unmöglich	Anwendung
37	Eingangsfehler (boolesche Karte)	Anwendung
38	Eingangsfehler (analoge Karte)	Anwendung
39	Eingangsfehler (Karte Typ Zeichenkette)	Anwendung
40	Ausgangsfehler (boolesche Variable)	Anwendung
41	Ausgangsfehler (analoge Variable)	Anwendung
42	Ausgangsfehler (Zeichenketten-Variable)	Anwendung
43	Verarbeitung "operate" nicht definiert (boolesche Variable)	Anwendung

44	Verarbeitung "operate" nicht definiert (analoge Variable)	Anwendung
45	Verarbeitung "operate" nicht definiert (Zeichenketten-Variable)	Anwendung
46	E/A-Karte kann nicht geöffnet werden	Anwendung
47	E/A-Karte kann nicht geschlossen werden	Anwendung
50	Neuschreiben verweigert (boolesche Ausgangsvariable)	Programm
51	Neuschreiben verweigert (analoge Ausgangsvariable)	Programm
52	Neuschreiben verweigert (Ausgangsvariable Typ Zeichenkette)	Programm
61	unbekannter Code für die "System"-Funktion	Programm
62	Überschreitung der Sampling-Periode	Programm
63	Anwenderfunktion nicht implementiert	Anwendung
64	durch null geteilte Ganzzahl	Programm
65	Umrechnungsfunktion nicht implementiert	Anwendung
66	Funktionsbaustein nicht implementiert	Anwendung
67	Standard-Funktion nicht implementiert	Anwendung
68	durch null geteilte Realzahl	Programm
69	ungültige Parameter für "operate"	Anwendung
72	Anwendungssymbole können nicht geändert werden	Anwendung
73	Aktualisierung verweigert: neue Deklarationen von booleschen Variablen	Anwendung
74	Aktualisierung verweigert: neue Deklarationen von analogen Variablen	Anwendung
75	Aktualisierung verweigert: neue Deklarationen von Timer-Variablen	Anwendung
76	Aktualisierung verweigert: neue Deklarationen von Zeichenketten-Variablen	Anwendung
77	Aktualisierung verweigert: kann die neue Anwendung nicht finden	Anwendung
> 100	spezifischer OEM-Fehlercode. Für zusätzliche Informationen wenden Sie sich an Ihren Lieferanten.	

Hierbei unterscheidet man zwischen 3 Arten von Fehlern:

– Systemfehler:
Solche Probleme sind höchstwahrscheinlich auf die Software oder Hardware zurückzuführen und nicht auf die Anwendungseinstellung oder die Programmausführung.
Versuchen Sie, Ihr Zielsystem rückzusetzen (Strom ausschalten) und versuchen Sie, andere Anwendungen auszuführen.
Fehler dieser Art sollten dem technischen ISaGRAF Unterstützungsdienst gemeldet werden.

– Anwendungsfehler:
Solche Probleme sind auf Anwendungsparameter, -größe oder -inhalt zurückzuführen.
Fehler dieser Art sollten nicht mehr auftreten, wenn eine bekannte und zuvor validierte Anwendung geladen wird. Wenn das Problem auch weiterhin auftritt, wird es zu den Systemfehlern gerechnet (siehe oben).

– Programmfehler:
Solche Fehler sind auf eine bestimmte Programmsequenz zurückzuführen.

Fehler dieser Art sollten nicht mehr auftreten, wenn die Anwendung im Schrittmodus gestartet oder wenn das kritische Programm gestoppt wird.

Fehlerbeschreibung:

1. nicht genug Speicherplatz, um die Datenbank zu speichern	***System***

Kein Speicherplatz verfügbar. Hardware überprüfen.

2. unzusammenhängende Datenbank (Motorola/Intel)	***Anwendung***

Die geladene oder gesicherte Anwendungsdatei ist falsch. Dieser Fehler tritt auf, wenn die Anwendung für INTEL generiert ist und auf MOTOROLA (und umgekehrt) geladen wird oder wenn die Datei modifiziert wurde.

3. nicht genug Speicher, um die Kommunikations-Mailbox zu installieren	***System***

Dieser Fehler wird vom Kommunikationstask generiert, wenn er Platz 3 für die Intertaskkommunikation nicht zuordnen kann.

4. Verknüpfung mit der Datenbank unmöglich	***System***

Dieser Fehler wird vom Kommunikationstask generiert, wenn er keinen Kernel mit der in seiner Befehlszeile spezifizierten Slave-Nummer in der Ausführung finden kann.

5. Time-Out während einer Anfrage an den Kernel	***System***

Der Kommunikationstask kann keine Anfrage an den Kernel senden. Der Kernel befindet sich wahrscheinlich nicht in der Ausführung oder ist beschäftigt.

6. Time-Out während einer Antwort vom Kernel	***System***

Der Kommunikationstask kann keine Antwort vom Kernel empfangen. Der Kernel befindet sich wahrscheinlich nicht in der Ausführung oder ist beschäftigt.

7. Installation der Kommunikation unmöglich	***System***

Diese Warnmeldung wird generiert, wenn die Kommunikationsschicht keine physische Verbindung initialisieren kann. Die Meldung wird auch angezeigt, wenn kein Kommunikationspfad spezifiziert ist. Dies verhindert nicht, daß das Zielsystem ordnungsgemäß ausgeführt wird, es kann aber nicht kommunizieren.

8. kann nicht-flüchtigen Variablen keinen Speicher zuteilen	***Anwendung***

ISaGRAF kann keine nicht-flüchtigen Variablen verwalten. Es kann zwei Gründe für dieses Problem geben:
- die als Parameter an ein Host-Zielsystem gegebene Zeichenkette weist Syntaxfehler auf
- die für jeden Block spezifizierte Speichergröße ist unzureichend

Überprüfen Sie die Syntax Ihres Parameters "flüchtige Variablen" und versuchen Sie es mit einer verminderten Anzahl an nicht-flüchtigen Variablen.

9. Anwendung gestoppt	***Anwendung***

Diese Meldung erscheint jedesmal, wenn die Anwendung vom Debugger aus gestoppt wird.

10. zu viele N- oder P-Aktionen gleichzeitig	***Anwendung***

Dieser Fehler tritt auf, wenn einer der Zielsystemzyklen zu viele gespeicherte Impulsaktionen oder zyklische Blöcke ausführen muß. Das Problem kann im Schrittmodus lokalisiert werden. Die Höchstzahl gleichzeitiger Aktionen ist 2 + 4 pro AS-Programm.

11. zu viele Setze-Aktionen gleichzeitig	***Anwendung***

Dieser Fehler tritt auf, wenn einer der Zielsystemzyklen zu viele Setze-Aktionen ausführen muß (wenn ein Schritt aktiv wird). Gleiche Vorgehensweise wie oben.

12. zu viele Rücksetze-Aktionen gleichzeitig	***Anwendung***

Dieser Fehler tritt auf, wenn einer der Zielsystemzyklen zu viele Rücksetze-Aktionen ausführen muß (wenn ein Schritt deaktiviert wird). Gleiche Vorgehensweise wie oben.

13. TIC-Anweisung unbekannt	***Anwendung***

Der Kernel hat einen Fehler im Anwendungscode eines Programms erkannt (genannt Target Independent Code). Dieses Problem kann zwei Gründe haben:
- ein externes Programm schreibt in den Anwendungscode. Versuchen Sie, das Problem im Schrittmodus zu lokalisieren und vergewissern Sie sich, daß alle E/A-Schnittstellen die richtigen Parameter haben.
- Ihr Zielsystem arbeitet mit einem reduzierten Satz an Anweisungen und Ihre Anwendung benutzt eine Anweisung oder einen Variablentyp, der nicht gestattet ist.

16. Antwort auf eine Leseanforderung unmöglich	***System***

Es wurde ein Kommunikationsfehler bei der Beantwortung einer spezifischen ISaGRAF Modbus Anfragefunktion Code 18 (Datei lesen) erkannt. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.

17. Antwort auf eine Schreibanforderung unmöglich	***System***

Es wurde ein Kommunikationsfehler bei der Beantwortung einer spezifischen ISaGRAF Modbus Anfragefunktion Code 17 (Datei schreiben) erkannt. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.

18. Antwort auf Session-Anforderung unmöglich	***System***

Es wurde ein Kommunikationsfehler bei der Beantwortung einer Debugger-Anfrage erkannt. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.

19. Antwort auf Modbus-Anforderung unmöglich	***System***

Es wurde ein Kommunikationsfehler bei der Beantwortung einer Modbus-Anfrage erkannt. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.

20. Antwort an Debugger unmöglich	***System***

Es wurde ein Kommunikationsfehler bei der Beantwortung einer Debugger-Anfrage erkannt. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.

21. Antwort an Debugger unmöglich	***System***

Es wurde ein Kommunikationsfehler bei der Beantwortung einer Debugger-Anfrage erkannt. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.

23. Anforderungs-Code unbekannt	***System***

Eine Debugger-Anfrage hat keinen Sinn.

24. Ethernet-Kommunikationsfehler	***System***

Diese Meldung erscheint jedesmal, wenn die Verbindung beim Schließen des Debuggers geschlossen wird: das System läuft OK. Ansonsten bedeutet die Meldung, daß ein Ethernet-Kommunikationsfehler erkannt wurde. Prüfen Sie den Anschluß und die Systemkonfiguration für Zielsystem und Master.
Eine zusätzliche Meldung kann lauten:
1: error while sending or receiving
2: error while creating the socket
3: error while binding or listening the socket
4: error while accepting a new client

25. Fehler der Kommunikationssynchronisierung	***System***

Schlechte Synchronisierung zwischen dem Kommunikationstask auf dem Zielsystem und dem Master. Prüfen Sie den Anschluß und die Systemkonfiguration (Kommunikationsparameter) für Zielsystem und Master.

28. nicht genug Speicher für die Anwendung	***System***

Kein Speicherplatz verfügbar. Prüfen Sie die Hardware hinsichtlich der Anwendungsgröße.

29. nicht genug Speicher für die Anwendungsaktualisierung	***System***

Kein Speicherplatz verfügbar. Prüfen Sie die Hardware hinsichtlich der Anwendungsgröße.

30. OEM-Schlüsselcode unbekannt	***Anwendung***

Die Anwendung benutzt eine Karte, dessen Herstellercode vom Zielsystem nicht erkannt wird. Prüfen Sie die E/A-Verdrahtung in der Workstation und benutzen Sie das Attribut "VIRTUELL",

um die falsche Karte zu lokalisieren. Es ist möglich, daß Ihre Workstation-Bibliothek der Version Ihres Zielsystems nicht entspricht.

31. Initialisierung einer booleschen Eingangskarte unmöglich	***Anwendung***

Die Initialisierung einer booleschen Eingangskarte ist mislungen. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Parameter Ihrer booleschen Eingangskarten.

32. Initialisierung einer analogen Eingangskarte unmöglich	***Anwendung***

Die Initialisierung einer analogen Eingangskarte ist mislungen. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Parameter Ihrer analogen Eingangskarten.

33. Initialisierung einer Eingangskarte vom Typ Zeichenkette unmöglich	***Anwendung***

Die Initialisierung einer Zeichenketten-Eingangskarte ist mislungen. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Parameter Ihrer Zeichenketten-Eingangskarten.

34. Initialisierung einer booleschen Ausgangskarte unmöglich	***Anwendung***

Die Initialisierung einer booleschen Ausgangskarte ist mislungen. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Parameter Ihrer booleschen Ausgangskarten.

35. Initialisierung einer analogen Ausgangskarte unmöglich	***Anwendung***

Die Initialisierung einer analogen Ausgangskarte ist mislungen. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Parameter Ihrer analogen Ausgangskarten.

36. Initialisierung einer Ausgangskarte vom Typ Zeichenkette unmöglich	***Anwendung***

Die Initialisierung einer Zeichenketten-Ausgangskarte ist mislungen. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Parameter Ihrer Zeichenketten-Ausgangskarten.

37. Eingangsfehler (boolesche Karte)	***Anwendung***

Bei der Aktualisierung einer booleschen Eingangskarte wurde ein Fehler erkannt. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Kartenparameter.

38. Eingangsfehler (analoge Karte)	***Anwendung***

Bei der Aktualisierung einer analogen Eingangskarte wurde ein Fehler erkannt. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Kartenparameter.

39. Eingangsfehler (Karte Typ Zeichenkette)	***Anwendung***

Bei der Aktualisierung einer Zeichenketten-Eingangskarte wurde ein Fehler erkannt. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Kartenparameter.

40. Ausgangsfehler (boolesche Variable)	***Anwendung***

Bei der Aktualisierung einer booleschen Ausgangsvariablen wurde ein Fehler erkannt. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Kartenparameter.

41. Ausgangsfehler (analoge Variable)	***Anwendung***

Bei der Aktualisierung einer analogen Ausgangsvariablen wurde ein Fehler erkannt. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Kartenparameter.

42. Ausgangsfehler (Zeichenketten-Variable)	***Anwendung***

Bei der Aktualisierung einer Zeichenketten-Ausgangsvariablen wurde ein Fehler erkannt. Prüfen Sie die E/A-Verdrahtung in der Workstation und die Kartenparameter.

43. Verarbeitung "operate" nicht definiert (boolesche Variable)	***Anwendung***

Bei einem OPERATE-Aufruf an eine boolesche Variable wurde ein Fehler erkannt. Prüfen Sie Ihre OPERATE-Parameter und die Gebrauchsanweisung der Karte.

44. Verarbeitung "operate" nicht definiert (analoge Variable)	***Anwendung***

Bei einem OPERATE-Aufruf an eine analoge Variable wurde ein Fehler erkannt. Prüfen Sie Ihre OPERATE-Parameter und die Gebrauchsanweisung der Karte.

45. Verarbeitung "operate" nicht definiert (Zeichenketten-Variable)	***Anwendung***

Bei einem OPERATE-Aufruf an eine Zeichenketten-Variable wurde ein Fehler erkannt. Prüfen Sie Ihre OPERATE-Parameter und die Gebrauchsanweisung der Karte.

46. E/A-Karte kann nicht geöffnet werden	***Anwendung***

Die Anwendung benutzt eine Kartenreferenz, die dem Zielsystem nicht bekannt ist. Prüfen Sie die E/A-Verdrahtung in der Workstation. Es ist möglich, daß Ihre Workstation-Bibliothek der Version Ihres Zielsystems nicht entspricht.

47. E/A-Karte kann nicht geschlossen werden	***Anwendung***

Die Anwendung benutzt eine Kartenreferenz, die dem Zielsystem nicht bekannt ist. Prüfen Sie die E/A-Verdrahtung in der Workstation.

50. Neuschreiben verweigert (boolesche Ausgangsvariable)	***Programm***

Zwei AS-Sequenzen schreiben dieselbe boolesche Ausgangsvariable im selben Zielsystem-Zyklus. Dies sollte vermieden werden, um riskantes Verhalten der E/As zu vermeiden. Im Falle eines solchen Konflikts wird die Priorität an das hierarchie-höchste Programm gegeben. Wenn die beiden AS-Programme auf derselben Ebene liegen, kann dies unvorhersehbare Folgen haben.

51. Neuschreiben verweigert (analoge Ausgangsvariable)	***Programm***

Zwei AS-Sequenzen schreiben dieselbe analoge Ausgangsvariable im selben Zielsystem-Zyklus. Siehe oben.

52. Neuschreiben verweigert (Ausgangsvariable Typ Zeichenkette)	***Programm***

Zwei AS-Sequenzen schreiben dieselbe Zeichenketten-Ausgangsvariable im selben Zielsystem-Zyklus. Siehe oben.

61. unbekannter Code für die "System"-Funktion	***Programm***

Ein Programm benutzt den SYSTEM-Aufruf mit einem ungültigen Code.

62. Überschreitung der Sampling-Periode	***Programm***

Die Zykluszeit des Zielsystems ist länger als im Workstation-Menü spezifiziert.

Auf einem Multitasking-System bedeutet dies, daß die Zentraleinheit nicht genug Zeit hat, einen Zyklus auszuführen, selbst wenn die "aktuelle Zykluszeit" kürzer als die spezifizierte Zeit ist.

Auf einem Monotasking-System bedeutet dies immer, daß in einem der Zielsystem-Zyklen zu viele Operationen erfolgen.

Es gibt mehrere Möglichkeiten, dieses Problem zu beheben:
- Reduzieren Sie die Anzahl der Operationen, die zum Zeitpunkt der Meldungsanzeige ausgeführt werden.
- Reduzieren Sie die Anzahl der Zugriffsberechtigungen und gültigen Transitionen, optimieren Sie komplexe Verarbeitungen, usw.
- Reduzieren Sie die CPU-Last anderer Tasks, um ISaGRAF mehr Zeit zur Verfügung zu stellen.
- Reduzieren Sie den Kommunikationsverkehr, um ISaGRAF mehr Zeit zur Verfügung zu stellen.
- Benutzen Sie die dynamische Zykluszeit-Modifizierung, um die Zykluszeit den verschiedenen Verarbeitungsabschnitten anzupassen.
- Setzen Sie die Zykluszeit auf null, um den ISaGRAF Kernel so schnell wie möglich laufen zu lassen, ohne daß er auf Zykluszeit-Überschreitungen geprüft wird.

63. Anwenderfunktion nicht implementiert	***Anwendung***

Ein Programm benutzt eine C-Funktion, die im Zielsystem unbekannt ist. Es ist möglich, daß Ihre Workstation-Bibliothek der Version Ihres Zielsystems nicht entspricht.

64. durch null geteilte Ganzzahl	***Programm***

Ein Programm versucht, einen ganzzahligen analogen Wert durch null zu teilen. Ihre Anwendung sollte das verhindern, denn es kann unvorhersehbare Auswirkungen haben.
In diesem Fall errechnet ISaGRAF den höchsten analogen Wert als Ergebnis.
Wenn der Operand negativ ist, wird das Ergebnis invertiert.

65. Umrechnungsfunktion nicht implementiert	***Anwendung***

Ein Programm benutzt eine C-Umrechnungsfunktion, die im Zielsystem unbekannt ist. Es ist möglich, daß Ihre Workstation-Bibliothek der Version Ihres Zielsystems nicht entspricht.
In diesem Fall rechnet ISaGRAF den Wert nicht um.

66. Funktionsbaustein nicht implementiert	***Anwendung***

Ein Programm benutzt einen C-Funktionsbaustein, der im Zielsystem unbekannt ist. Es ist möglich, daß Ihre Workstation-Bibliothek der Version Ihres Zielsystems nicht entspricht.

67. Standard-Funktion nicht implementiert	***Anwendung***

Ein Programm benutzt einen Funktionsbaustein, der im Zielsystem unbekannt ist, obwohl er auf den meisten Zielsystemen verfügbar sein sollte. Fragen Sie Ihren Lieferanten.

68. durch null geteilte Realzahl	***Programm***

Ein Programm versucht, einen realen analogen Wert durch null zu teilen. Ihre Anwendung sollte das verhindern, denn es kann unvorhersehbare Auswirkungen haben.
In diesem Fall errechnet ISaGRAF den höchsten realen analogen Wert als Ergebnis.
Wenn der Operand negativ ist, wird das Ergebnis invertiert.

69. ungültige Parameter für "operate"	***Anwendung***

Ihre Anwendung benutzt einen OPERATE-Aufruf mit falschen Parametern. Dies wird gewöhnlich vom Compiler gefiltert. Es kann sich um einen Timer-Parameter oder um eine Variable handeln, die kein Eingang oder Ausgang ist.

72. die Anwendungssymbole können nicht geändert werden	***Anwendung***

Bei dem Versuch einer Anwendungsaktualisierung kann die modifizierte Anwendung nicht starten, weil die Symbole sich geändert haben. Im Vergleich zur aktuellen Anwendung wurden möglicherweise eine oder mehrere Variablen oder Funktionsbaustein-Instancen hinzugefügt, entfernt oder modifiziert.

73. Aktualisierung verweigert: neue Deklarationen von booleschen Variablen	***Anwendung***

Die modifizierte Anwendung kann nicht starten, weil im Vergleich zur aktuellen Anwendung boolesche Variablen hinzugefügt oder entfernt wurden.

74. Aktualisierung verweigert: neue Deklarationen von analogen Variablen	***Anwendung***

Die modifizierte Anwendung kann nicht starten, weil im Vergleich zur aktuellen Anwendung analoge Variablen hinzugefügt oder entfernt wurden.

75. Aktualisierung verweigert: neue Deklarationen von Timer-Variablen	***Anwendung***

Die modifizierte Anwendung kann nicht starten, weil im Vergleich zur aktuellen Anwendung Timer-Variablen hinzugefügt oder entfernt wurden.

76. Aktualisierung verweigert: neue Deklarationen von Zeichenketten-Variablen	***Anwendung***

Die modifizierte Anwendung kann nicht starten, weil im Vergleich zur aktuellen Anwendung Zeichenketten-Variablen hinzugefügt oder entfernt wurden.

77. Aktualisierung verweigert: kann die neue Anwendung nicht finden	***Anwendung***

Die modifizierte Anwendung kann im Speicher nicht gefunden werden. Die Ursache kann ein fehlerhafter Ladevorgang sein.

D. Glossar

Ablaufsprache Grafische Sprache: Der Prozeß wird in Schritte (stabile Zustände) zerlegt, die mit Transitionen verbunden sind. Den Schritten werden Aktionen zugeordnet. Die Überschreitung der Transitionen unterliegt Booleschen Bedingungen.

Abschnitt Gruppe von Programmen, die bei der Ausführung den gleichen dynamischen Regeln unterliegen.

Aktion (FD) Symbol im Flußdiagramm. Eine Aktion kennzeichnet eine Liste von Anweisungen, welche ausgeführt werden, wenn der dynamische Datenfluß das betreffende Aktionssymbol antrifft.

Aktion Liste an Anweisungen oder Zuweisungen, die ausgeführt werden, wenn der Schritt eines AS-Programms aktiv ist.

Aktuelles Ergebnis (AWL) Ergebnis einer AWL-Anweisung. Das aktuelle Ergebnis kann von einer Anweisung geändert werden und kann benutzt werden, um eine Variable zu forcieren.

Allgemein Geltungsbereich von Definitionen. Solche Objekte können von allen Programmen sämtlicher Projekte benutzt werden.

Analog Variablentyp. Die Variablen können kontinuierlich, real oder ganzzahlig sein.

Anfangsabschnitt Gruppe zyklischer Programme, die am Anfang eines jeden Zielsystemzyklus ausgeführt werden.

Anfangsschritt Erster Schritt im Rumpf eines Makro-Schritts. Anfangsschritte sind nicht mit vorangehenden Transitionen verknüpft.

Anschluß (FD) Grafisches Element im Flußdiagramm, welches eine Verbindung von einem Punkt im Flußdiagramm zu einer FD-Aktion oder einem FD-Test darstellt. Die grafische Darstellung eines Sprungs ist ein kleiner Kreis mit der Referenznummer des Zielelements.

Anweisung (ST) Vollständige ST-Operation.

Anweisung Elementare Operation eines AWL-Programms, die auf einer Zeile eingegeben wird.

Anweisungsliste Niedrige Textsprache, die als sequentielle Liste elementarer Anweisungen eingegeben wird.

AS Ablaufsprache.

AS-Sohnprogramm AS-Programm, das von einem anderen Programm, genannt AS-

	Vaterprogramm, gesteuert wird.
AS-Vaterprogramm	AS-Programm, das andere AS-Programme, genannt AS-Sohnprogramme, steuert.
Attribut	Variablenklasse. Verfügbare Attribute sind intern, Ein- oder Ausgang.
Ausdruck	Gruppe von Operatoren und Bezeichnern, welche die Bewertung eines Werts darstellt.
Ausführungsfehler	Anwendungsfehler, der bei der Ausführung vom ISaGRAF Zielsystem erkannt wurde.
Ausgang	Variablenattribut. Solche Variablen sind mit Ausgangseinrichtungen des Zielsystems verbunden.
AWL	Anweisungsliste.
Bedingung (einer Transition)	Boolescher Ausdruck, der einer AS-Transition zugeordnet ist. Die Transition kann nur dann überschritten werden, wenn die Bedingung TRUE ist.
Bezeichner	Einmaliges Wort, das benutzt wird, um eine Variable oder einen konstanten Ausdruck in der Programmierung darzustellen.
Bibliothek	Hardware- und Software- Resourcen, welche unmittelbar in einer Anwendung benutzt werden können.
Boolesch	Variablentyp. Diese Variablen können ausschließlich die Zustände "true" oder "false" annehmen.
Boolesche Aktion	AS-Aktion: Einer Booleschen Variablen wird das Aktivitätssignal eines Schritts zugewiesen.
C-Funktion	In der "C"-Sprache geschriebene Funktion, die von den (in anderen Sprachen geschriebenen) ISaGRAF Programmen synchron aufgerufen wird. C-Funktionen werden von CJ International geliefert oder vom Anwender entwickelt.
C-Quellcode	Textdatei, die den Quellcode einer Funktion oder einer Umrechnungsfunktion enthält.
C-Quellcode-vorsatz	Textdatei, welche die "C"-Definitionen und Typen enthält, die für die Programmierung einer C-Funktion oder einer Umrechnungsfunktion erforderlich sind.
Crossreferenzen	(Auch Querverweise genannt.) Von der ISaGRAF Workstation errechnete Daten über das Datenverzeichnis eines Projekts und die Anwendung dieser Daten in den Projektprogrammen.
C-Sprache	Hohe Textsprache für die Beschreibung von Computeroperationen, wie z.B. C-Funktionen und Umrechnungsfunktionen.

Datenverzeichnis Alle internen, Ein- und Ausgangsvariablen und Definitionen, die in den Programmen eines Projekts benutzt werden.

Definition Einmaliger Bezeichner, der benutzt wird, um einen Ausdruck in einem Programm zu ersetzen.

E/A-Kanal Einzelner Anschlußpunkt einer E/A-Karte. Ein E/A-Kanal kann eine E/A-Variable aufnehmen.

E/A-Karte Hardware-Resource. Eine E/A-Karte charakterisiert sich durch einen Typ und eine Richtung (Eingang oder Ausgang). Die E/A-Kartenparameter werden in der IsaGRAF Bibliothek beschrieben.

E/A-Variable An eine E/A-Einrichtung angeschlossene Variable. Eine E/A-Variable wird an den Kanal einer E/A-Karte angeschlossen.

E/A-Verdrahtung Definition der Verknüpfungen zwischen den Variablen einer Anwendung und den Kanälen der im Zielsystem vorhandenen E/A-Karten.

Ebene 1 der AS Hauptbeschreibung eines AS-Programms. Die Ebene 1 beinhaltet das Netzwerk (Schritte und Transitionen), sowie die zugeordneten Kommentare.

Ebene 2 der AS Detaillierte Beschreibung eines AS-Programms. Die Ebene 2 umfaßt die Programmierung der Aktionen in den Schritten sowie der Bedingungen, welche den Transitionen zugeordnet sind.

Echtzeitmodus Normaler Ausführungsmodus: Die Zielsystemzyklen werden von der programmierten Zykluszeit ausgelöst.

Eingang Variablenattribut. Eingangsvariablen sind mit Eingangseinrichtungen verknüpft.

Endabschnitt Gruppe zyklischer Programme, die am Ende eines jeden Zielsystemzyklus ausgeführt werden.

Endschritt Letzter Schritt im Rumpf eines Makro-Schritts. Endschritte sind nicht mit nachfolgenden Transitionen verknüpft.

Entscheidung (FD) (auch Test genannt.) Ein Symbol im Flußdiagramm, welches einem Booleschen Ausdruck zugeordnet ist. Der Zustand des Ausdrucks bedingt, ob der Datenfluß zu einem YES-Symbol oder zu einem NO-Symbol (Ausgang) geleitet wird.

FBS Funktionsbaustein-Sprache.

FD Abkürzung für "Flußdiagramm".

Flanke Veränderung einer Booleschen Variablen. Eine positive Flanke ist ein Übergang von "false" auf "true". Eine negative Flanke ist ein Übergang von "true" auf "false".

Flußdiagramm Grafische Sprache zur Beschreibung eines Datenflusses. Das Diagramm besteht aus Aktionen, die ausgeführt werden, und Tests (Entscheidungen), welche die Auswahl zwischen verschiedenen Wegen im Datenfluß bedingen. Die Flußdiagramm-Sprache ermöglicht die Erstellung von Schleifen, die in aufeinanderfolgenden Zyklen ausgeführt werden.

Funktionsbaustein Grafisches Element der FBS-Sprache, welches eine elementare, in den ISaGRAF Bibliotheken definierte Funktion darstellt.

Funktionsbaustein-Sprache Grafische Sprache, in der Gleichungen mit den Standard-Funktionsbausteinen der ISaGRAF Bibliotheken erstellt und im Diagramm verknüpft werden.

Ganzzahlig Analoge Variablen, die in einem vorzeichenbehafteten ganzzahligen 32 Bit-Format gespeichert werden.

Geltungsbereich Programmgruppe, die ein Objekt benutzen kann. Die in ISaGRAF vordefinierten Geltungsbereiche sind: allgemein, global und lokal.

Global Geltungsbereich von Variablen und Definitionen. Globale Objekte können von allen Programmen eines Projekts benutzt werden.

Gültigkeit einer Transition Attribut einer Transition. Eine Transition ist gültig, wenn alle der Transition unmittelbar vorangehenden Schritte aktiv sind.

Haltepunkt Marke, die beim Debuggen vom Benutzer auf einen AS-Schritt oder eine AS-Transition gesetzt werden kann. Das Zielsystem stoppt, wenn eine AS-Zugriffsberechtigung einen Haltepunkt antrifft.

Hauptprogramm Programm, das die höchste Priorität in der hierarchischen Baumstruktur hat. Hauptprogramme werden vom System aktiviert.

Hierarchie Architektur eines Projekts, das aus mehreren Programmen besteht. Die hierarchische Baumstruktur veranschaulicht die Verknüpfungen zwischen Vater- und Sohnprogrammen.

Impulsaktion AS-Aktion: Anweisungsliste, die ein einziges Mal ausgeführt wird, wenn der betreffende Schritt aktiviert wird.

Initale Situation Initialisierungsschritte eines AS-Programms, die den Kontext des Programms beim Programmstart darstellen.

Initialisierungs-schritt Schritt eines AS-Programms, der vom System beim Programmstart aktiviert wird.

Intern Attribut einer Variablen, die nicht mit Ein- oder Ausgangseinrichtungen verknüpft ist.

Kommentar In ein Programm eingefügter Text, der die Programmausführung nicht beinflußt.

Kommentar (AS) Einem AS-Schritt oder einer AS-Transition zugeordneter Text, der die Programmausführung nicht beinflußt.

Konstanter Ausdruck Literaler Ausdruck, der benutzt wird, um einen konstanten Wert zu beschreiben. Ein konstanter Ausdruck ist stets typ-dediziert.

Kontakt Grafisches Element im KOP-Diagramm, welches den Zustand einer Eingangsvariablen darstellt.

Kontaktplan Grafische Sprache, in der Kontakte und Spulen miteinander kombiniert werden, um Boolesche Gleichungen darzustellen.

KOP Kontaktplan-Sprache.

Lokal Geltungsbereich von Variablen und Definitionen. Solche Objekte können nur von einem einzigen Programm eines Projekts benutzt werden.

Makroschritt Grafisches Element der AS-Sprache. Ein Makroschritt ist eine einmalige Gruppe von Schritten und Transitionen, welche durch ein einmaliges Symbol im Hauptnetzwerk dargestellt und separat beschrieben wird.

Marke (AWL) An den Anfang einer AWL-Anweisungszeile gesetzter Bezeichner, der die Anweisung kennzeichnet und als Operand für Sprungoperationen benutzt werden kann.

Matrix Logische Aufteilung eines grafischen Eingabefelds in rechteckige Zellen während der Bearbeitung eines Programms in einer grafischen Sprache.

Modbus Protokoll vom Typ Master-Slave. Ein ISaGRAF Zielsystem kann in einer Architektur Modbus-untergeordnet (an ein externes System, z.B. einen Supervisor angeschlossen) sein.

Modifizierer (AWL) Einzelnes Zeichen, das an das Ende eines AWL-Schlüsselworts gesetzt wird und die Bedeutung der betreffenden Operation modifiziert.

Netzwerkadresse Optionale hexadezimale Adresse, die für jede Variable frei definiert wird. Diese Adresse wird vom Modbus-Protokoll benutzt, wenn das Zielsystem an ein externes System angeschlossen ist.

Nicht-gespeicherte Aktion AS-Aktion: Anweisungsliste, die bei jedem Zyklus ausgeführt wird, wenn der betreffende Schritt aktiv ist.

OEM-Parameter (E/A-Karte) Parameter einer E/A-Karte, der vom Kartenhersteller definiert wird. Dabei kann es sich um einen konstanten Wert handeln oder um einen veränderbaren Parameter, der bei der E/A-Verdrahtung für jede Anwendung neu eingegeben wird.

OEM-Schlüsselcode Hexadezimalcode auf 16 Bits, der den Hersteller einer E/A-Karte der ISaGRAF Bibliothek kennzeichnet.

(E/A-Karte)

Operand (AWL) Variabler oder konstanter Ausdruck, der von einer elementaren AWL-Anweisung verarbeitet wird.

Operation (AWL) Grundlegende Anweisung der AWL-Sprache. In einer Anweisung wird einem Operanden gewöhnlich eine Operation zugeordnet.

Parameter (E/A-Karte) Konstanter oder benutzerdefinierbarer Parameter einer Standard-E/A-Karte. Ein programmierbarer Parameter wird bei der E/A-Verdrahtung eingegeben.

Parameter (C-Funktion) Eingangswert einer "C"-Funktion. Ein Parameter wird durch einen Typ definiert.

Programm Grundlegende Programmiereinheit eines Projekts. Ein Programm wird in einer einzigen Sprache geschrieben und hat einen bestimmten Platz in der hierarchischen Baumstruktur des Projekts.

Programmprotokoll Textdatei, das Notizen bezüglich der Entwicklung eines Programms enthält. Jede Notiz wird mit Eingabedatum gespeichert.

Projekt Programmierbereich, der sämtliche Informationen (Programme, Variablen, Zielcode...) einer ISaGRAF Anwendung zusammenfaßt.

Reale Karte Echte E/A-Karte, die an die E/A-Einrichtungen eines Zielsystems angeschlossen ist.

Real Analoge Variablen, die in einem Gleitpunkt-IEEE-Format auf 32 Bits (einfache Präzision) gespeichert werden.

Referenznummer (AS) Dezimalzahl (zwischen 1 und 65535), die einen Schritt oder eine Transition in einem AS-Programm kennzeichnet.

Register (AWL) Aktuelles Ergebnis einer AWL-Sequenz.

Rückgabewert eines Unterprogramms Von einem Unterprogramm nach beendeter Ausführung zurückgegebener Wert. Der Rückgabewert wird in den Operationen des aufrufenden Programms benutzt.

Schlüsselwort Reserviertes Wort einer Sprache.

Schrittaktivität Attribut eines Schritts, der mit einer AS-Zugriffsberechtigung markiert ist. Die Aktionen des Schritts werden entsprechend seiner Aktivität ausgeführt.

Schritt Grafisches Element der AS-Sprache. Ein Schritt kennzeichnet einen stabilen Prozeßzustand und wird als Quadrat dargestellt. Schritte werden mit einer Referenznummer gekennzeichnet. Die Schrittaktivität steuert die Ausführung der betreffenden Aktionen.

Schrittmodus Ausführungsmodus: In diesem Modus werden die Zyklen schrittweise

	ausgeführt, entsprechend der Benutzerbefehle beim Debuggen.
Sequentieller Abschnitt	Gruppe von Programmen eines Projekts. Die Ausführung dieser Programme unterliegt den dynamischen Ablaufregeln der AS-Sprache.
Sprung zu einem Schritt	Grafisches Element in der AS-Sprache, welches die Verbindung einer Transition mit einem Schritt darstellt. Das grafische Symbol eines Sprungs ist ein Pfeil, der mit der Nummer des Zielschritts gekennzeichnet ist.
Spule	Grafisches Element im KOP-Diagramm, das benutzt wird, um die Zuweisung einer Ausgangsvariablen darzustellen.
ST	Strukturierter Text.
String (Zeichenfolge)	Variablentyp. Solche Variablen enthalten unterschiedlich lange Zeichenfolgen.
Stromschiene	Vertikale Linien, die einen Kontaktplan links und rechts begrenzen.
Strukturierter Text	Hohe strukturierte Textsprache, in der Zuweisungen, hohe Strukturen (z.B. If/Then/Else) und Funktionsaufrufe miteinander kombiniert werden.
Technisches Datenblatt	Gebrauchsanleitung für ein Element der ISaGRAF Bibliotheken (C-Funktion oder -Funktionsbaustein, Umrechnungsfunktion oder E/A-Karte). Das technische Datenblatt wird vom Designer des Elements erstellt.
Test (FD)	(Auch Entscheidung genannt.) Ein Symbol im Flußdiagramm, welches einem Booleschen Ausdruck zugeordnet ist. Der Zustand des Ausdrucks bedingt, ob der Datenfluß zu einem YES-Symbol oder zu einem NO-Symbol (Ausgang) geleitet wird.
Timer	Variablentyp. Diese Variablen enthalten Zeitwerte und können bei der Ausführung automatisch von ISaGRAF System aktualisiert werden.
Toolbox	Kleines Fenster in einem grafischen Bearbeitungsfenster, welches die Schaltflächen für die Auswahl der grafischen Elemente enthält.
Transition	Grundlegendes grafisches Element der AS-Sprache. Eine Transition stellt die Bedingung zwischen verschiedenen AS-Schritten dar. Sie wird mit einer Nummer gekennzeichnet. Der Transition wird ein Boolescher Ausdruck zugeordnet.
Trennzeichen	Spezialzeichen (oder Zeichengruppe) zum Trennen von zwei Bezeichnern in einer Textsprache. Ein Trennzeichen kann eine Operation darstellen.
Typ	Variablen von gleichem Format. Vordefinierte Variablentypen sind: Boolesch, analog, Timer und Zeichenfolge.

Überschreitung einer Transition Operation bei der Ausführung eines AS-Programms. Alle Zugriffsberechtigungen in den vorangehenden Schritten werden entfernt. In allen nachfolgenden Schritten werden Zugriffsberechtigungen erstellt.

Umrechnung Filter, der einer analogen Ein- oder Ausgangsvariablen zugeordnet wird. Die Umrechnung erfolgt automatisch bei jeder Erfassung oder Aktualisierung der E/A-Variablen.

Umrechnungs-funktion In der "C"-Sprache geschriebene Funktion, die eine Umrechnung beschreibt. Umrechnungsfunktionen können allen analogen Ein- oder Ausgangsvariablen zugeordnet werden.

Umrechnungs-tabelle Gruppe von Punkten, die eine lineare Umrechnung in Segmenten definiert. Umrechnungstabellen können auf alle analogen Ein- oder Ausgangsvariablen angewendet werden.

Unterprogramm In einer nicht-AS-Sprache geschriebenes Programm, das von einem anderen, sogenannten Vaterprogramm aufgerufen wird.

Variable Einmalige Darstellung einer elementaren Dateneinheit, die von den Programmen eines Projekts verarbeitet wird.

Vaterprogramm In einer beliebigen Sprache geschriebenes Programm, das ein anderes Programm (nicht AS), welches Unterprogramm genannt wird, steuert (aufruft).

Verriegelte E/A Ein- oder Ausgangsvariable, deren Anschluß an die entsprechende E/A-Einrichtung mit dem Debugger-Befehl "Verriegeln" unterbrochen wurde.

Verzögerte Operation (AWL) Operation in einem AWL-Programm, die erst später im Programm ausgeführt wird, wenn das Zeichen "(" angetroffen wird.

Virtuelle Karte E/A-Karte, die nicht wirklich an eine E/A-Einrichtung des Zielsystems angeschlossen ist.

Zeichenfolge Zeichengruppe, die in einer Zeichenfolgen-Variablen gespeichert wird.

Zelle Grundelement in der Bearbeitungsmatrix der grafischen Sprachen (AS, FBS oder KOP).

Zielsystem ISaGRAF Zielsteuerung, welche die ISaGRAF Kernelsoftware unterstützt.

Zielsystemzyklus Satz an Operationen, die jedesmal ausgeführt werden, wenn das ISaGRAF Zielsystem aktiviert wird. Die Zyklen werden gemäß einer programmierbaren Zykluszeit ausgelöst.

Zugriffs-berechtigung (AS) Grafische Markierung, die benutzt wird, um die Aktivität eines AS-Schritts aufzuzeigen.

Zyklisch Attribut eines Programms, das in jedem Zyklus ausgeführt wird.

Zykluszeit Dauer eines Zielsystemzyklus.

E. Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

Ü

V

W

X

Z