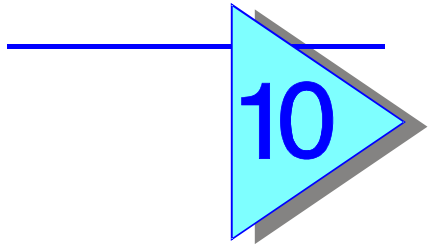


Blockorientierte Simulation mit BORIS

Übersicht

Anwendungsbereiche von BORIS

Das blockorientierte Simulationssystem BORIS ermöglicht die Simulation nahezu beliebig strukturierter dynamischer Systeme und eignet sich in Verbindung mit der Prozessschnittstelle und der optionalen C-Code-Generierung damit gleichermaßen für die Anwendungsbereiche

- Messwerterfassung und Signalanalyse,
- Regelkreisanalyse und -synthese,
- Systemoptimierung,
- Digitaltechnik,
- Konfigurierung und Parametrierung von Hardwarebaugruppen.

Neben den bekannten konventionellen Systemen können insbesondere auch Systeme mit Fuzzy- und/oder Neuro-Komponenten verarbeitet werden.

Besonders anspruchsvolle und komplexe Animationen und Prozessvisualisierungen lassen sich mit dem separat erhältlichen *Flexible Animation Builder* realisieren.

Einige Leistungsmerkmale von BORIS

Einige der wesentlichen Leistungsmerkmale von BORIS (je nach Ausbaustufe der erworbenen Version):

- Umfangreiche Systembibliothek:
  - Signalgeneratoren (Dreieck, Rechteck, Sinus, Impuls, Rauschen, div. Testfunktionen)
  - Lineare Standardglieder (PT_1, PT_2, ...)
  - Lineare Übertragungsglieder höherer Ordnung

- Nichtlineare Kennlinienglieder; frei definierbare Kennlinien; algebraische Funktionen
- Lineare und nichtlineare Standard-Reglerkomponenten
- Fuzzy Controller und Neuronale Netze
- Anbindung benutzerdefinierter Funktionsblöcke über DLLs
- Verschiedene Ausgabeblöcke (Zeitverläufe, Trajektorienverläufe, analoge und digitale Anzeigeinstrumente, Oszillograph, Statusanzeigen)
- Aktionsblöcke zum interaktiven Eingriff in den Simulationsablauf (z. B. Schalter, Potentiometer)
- Ein- und Ausgabe von Signalverläufen aus bzw. in Dateien
- Spektralanalyse über Fast-Fourier-Transformation
- Statistik-Funktionen
- Digitale Bausteine (z. B. Logikgatter und Flip-Flops)
- Kommunikations-Bausteine (z. B. DDE-, TCP/IP- und OPC-Blöcke)

- Definition hierarchischer Makros (Superblöcke)
- Anzahl der Systemblöcke nur durch den Hauptspeicher des Rechners begrenzt
- Beliebige Plazierbarkeit von Systemblöcken; nahezu beliebig große, scrollfähige Arbeitsfläche
- Verschiedene Integrationsverfahren
- Automatischer oder halbautomatischer Entwurf von PID-Reglern anhand von Einstellregeln
- Numerische Parameteroptimierung auf der Basis von Evolutionsstrategien
- Prozessankopplung über A/D- und D/A-Wandlerkarten, serielle Messmodule, Prozessleitsysteme etc.
- Überführung beliebiger Systemstrukturen in ANSI-C-Code durch den leistungsfähigen AutoCode-Generator

Im nachfolgenden Kapitel soll zunächst anhand eines ausführlichen Beispiels eine Einführung in die Arbeit mit BORIS erfolgen.

Eine erste Simulation mit BORIS

Ein einfaches Beispiel

Die Benutzeroberfläche von BORIS orientiert sich im wesentlichen am Windows-Standard sowie den WinFACT-typischen Konventionen. Im folgenden wollen wir uns in die grundsätzliche Handhabung des Programms anhand eines einfachen Beispiels einarbeiten.

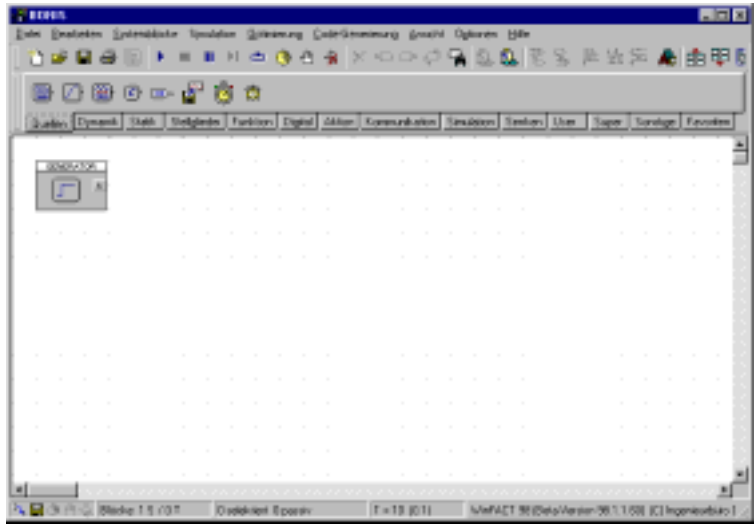
Das im folgenden beschriebene Beispiel befindet sich unter dem Namen DEMO1.BSY im Beispiel-Verzeichnis.

Wir wollen eine lineare Regelstrecke 2. Ordnung mit der Verstärkung $K = 2$ und den Zeitkonstanten $T_1 = 1$ und $T_2 = 2$ betrachten. Uns interessiert zunächst die Antwort $y(t)$ des Systems auf eine sinusförmige Eingangsgröße $u(t) = u_0 \sin \omega t$ mit der Amplitude $u_0 = 1$ und der Kreisfrequenz $\omega = 3$. Wie lässt sich dieses Problem mit BORIS lösen?

Betrachten wir zunächst das Anwendungshauptfenster von BORIS, wie es sich uns direkt nach dem Aufruf des Programms darstellt. Wir erkennen am oberen Rand zwei horizontale Toolbars. Die untere, die sogenannte *Systemblock-Toolbar* ermöglicht den direkten Zugriff auf alle Systemblöcke und ist in verschiedene Blockgruppen (*Palettenseiten*) aufgeteilt, die über die Register aktiviert werden können.

Systemblock-Toolbar von BORIS

Wir benötigen zunächst einen *Signalgenerator* zur Erzeugung unseres Eingangssignals. Diesen erhalten wir durch Anklicken der ersten Schaltfläche der Palettenseiten *Quellen*. Der Generator erscheint daraufhin in der linken oberen Ecke des Zeichenfensters. Das Einfügen des Systemblocks wird weiterhin in der Statuszeile des Hauptfensters quittiert.



Hauptfenster von BORIS nach dem Einfügen des Signalgenerators

Als nächstes wollen wir die Regelstrecke selbst einfügen. Dazu betätigen wir die mit *PT1T2* beschriftete Schaltfläche der Palette *Dynamik*. Der Block erscheint daraufhin rechts neben dem Generator.

Verschieben von Blöcken

Wir wollen den Block noch etwas nach rechts verschieben. Dazu aktivieren wir den Block durch Anklicken mit der linken Maustaste. Die Aktivierung ist am invertierten Blocktitel erkennbar. Bei festgehaltener linker Maustaste können wir den Block jetzt an der gewünschten Position plazieren.

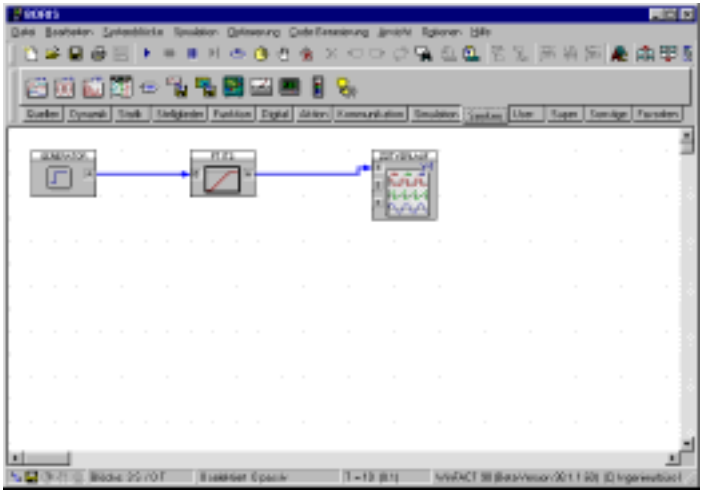
Alternativ lassen sich einzelne Blöcke auch per Drag & Drop einfügen. Dazu wird die linke Maustaste nach dem Anklicken der Block-Schaltfläche festgehalten; der Block kann dann auf dem Arbeitsblatt direkt in die gewünschte Position gebracht und dort „fallengelassen“ werden.

Nun benötigen wir noch einen Ausgabeblock zur späteren Anzeige der Simulationsergebnisse. Wir wollen dazu einen Block vom Typ *Zeitverlauf* wählen, den wir jetzt wieder über die Toolbar (Palettenseite *Senken*) einfügen und im Anschluss daran geeignet plazieren. Gleichzeitig mit der Darstellung des Blocksymbols wird ein zusätzliches Fenster geöffnet, das zunächst zum Symbol verkleinert ist. Dieses Anzeigefenster enthält später die eigentliche Simulationsgrafik.

Als nächstes wollen wir die notwendigen Verbindungen ziehen. Dabei gilt:

*Verbindungen werden **vom Ausgang zum Eingang** gezogen!*

Wir klicken daher zunächst den Ausgang unseres Signalgenerators - das mit "A" beschriftete, erhöht dargestellte Feld - mit der linken Maustaste an. Der Mauszeiger wechselt daraufhin seine Form und stellt - genügend Phantasie beim Anwender vorausgesetzt - einen stilisierten Lötkolben dar. Wir setzen den Cursor danach in das Eingangsfeld "E" unserer Regelstrecke - der Mauszeiger wechselt seine Farbe auf schwarz und zeigt ein Fadenkreuz - und betätigen nochmals die linke Maustaste. Die Verbindung wird daraufhin automatisch eingezeichnet. Auf die gleiche Weise verbinden wir den Ausgang der Strecke mit dem Eingang 1 des Zeitverlauf-Blocks.



Hauptfenster nach Einfügen aller Blöcke und Ziehen der Verbindungen

Parametrierung der Blöcke

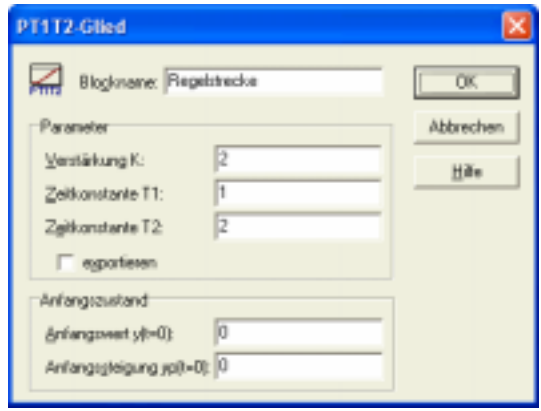
Die Struktur unseres Systems liegt nun fest. Bevor die Simulation gestartet werden kann, müssen wir unsere Blöcke natürlich noch parametrieren. Zunächst wollen wir dies für den Generator tun. Der einfachste Weg dazu besteht in einem Doppelklick mit der linken Maustaste auf den Systemblock. Es erscheint dann der zugehörige Eingabedialog, in dem wir die Änderungen vornehmen können. Wir müssen für unser Beispiel folgende Änderungen vornehmen, die in untenstehendem Dialog bereits eingetragen wurden:

- In der Schaltergruppe *Typ* wählen wir *Sinus* an.
- Im Feld *Kreisfrequenz* tragen wir den Wert 3 ein.

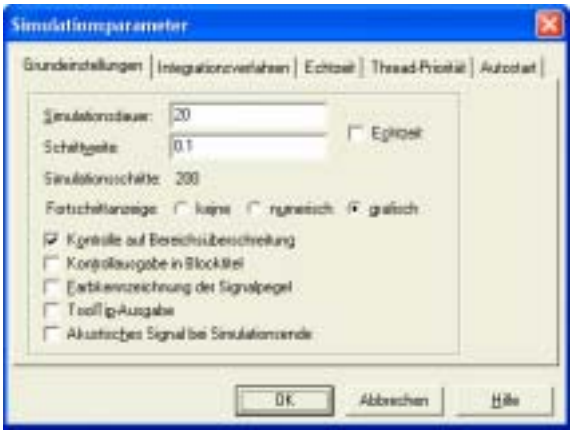
Nach Durchführung der Änderungen können wir die Einstellungen über die Schaltfläche *Test* austesten.

Parameterdialog für Generator

Auf die gleiche Weise parametrieren wir die Regelstrecke. Zur Verdeutlichung ändern wir hier zusätzlich den Blocknamen auf *Regelstrecke.* Zum Abschluss müssen wir die Simulationsparameter festlegen. Dazu wählen wir das Uhrensymbol in der oberen horizontalen Toolbar, der sogenannten *System-Toolbar* und gelangen dadurch in den entsprechenden Eingabedialog. Wir wählen eine Simulationsdauer von 20 und eine Simulationsschrittweite von 0.1. Aus diesen Einstellungen resultieren insgesamt 200 Simulationsschritte.

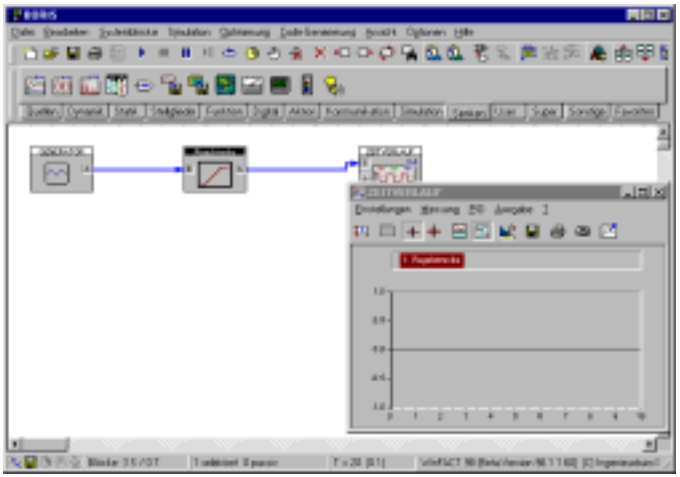


Parametrierung der Regelstrecke



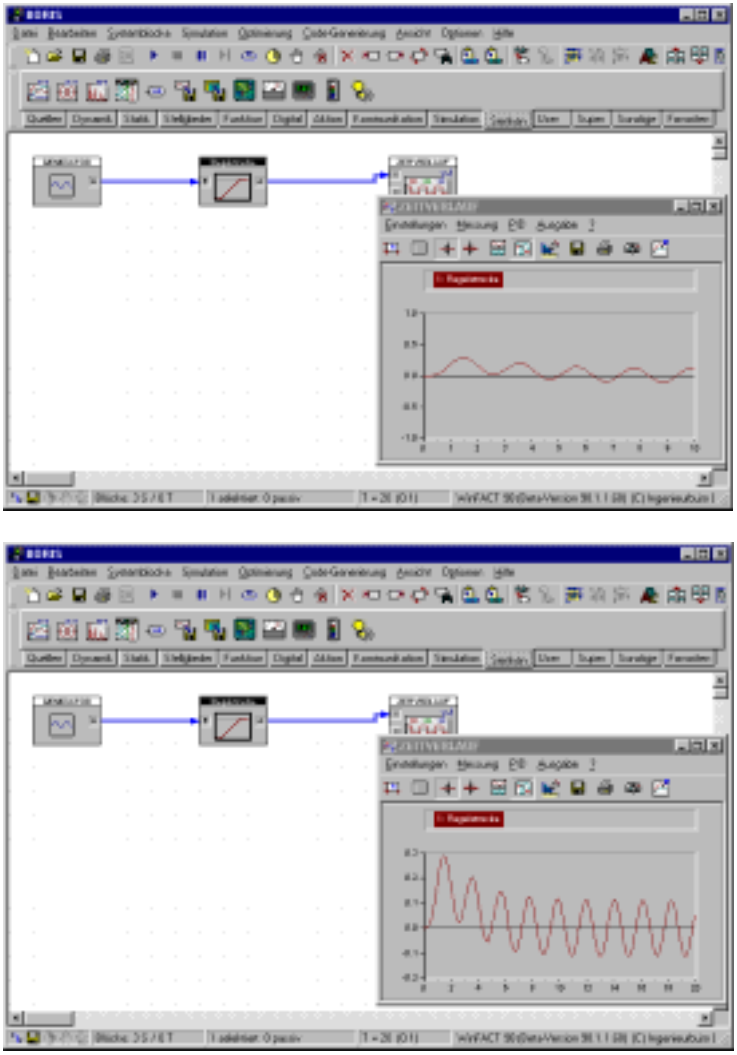
Einstellung der Simulationsparameter

Unser System ist jetzt simulationsbereit. Um den Simulationsverlauf bereits während der Simulation verfolgen zu können, holen wir das Zeitverlauf-Anzeigefenster durch einen Doppelklick aus der Symboldarstellung zurück. Es wird daraufhin in Normalgröße dargestellt, enthält allerdings natürlich noch keine Ergebnisse. Wir können das Fenster jetzt beliebig vergrößern und verkleinern. Da wir die Simulationsdauer auf 20 erhöht haben, müssen wir die Zeitachse anpassen. Dies können wir über die Menüoption EINSTELLUNGEN des Anzeigefensters bewerkstelligen.



Zeitverlauf-Anzeige in Normalgröße

Der Start der Simulation bedarf nunmehr nur noch eines einzigen Tastendrucks auf den Einfachpfeil in der System-Toolbar. Der Fortgang wird im Statusfenster des Hauptfensters und natürlich auch im Zeitverlauf-Anzeigefenster angezeigt. Nach Ende der Simulation stellen wir fest, dass die Skalierung der Amplitudenwerte innerhalb des Zeitverlauf-Anzeigefensters recht ungünstig ist. Dem können wir durch eine Betätigung der linken Schaltfläche in der Toolbar des Fensters abhelfen. Daraufhin erfolgt ein automatisches Neuzeichnen der Kurve mit optimaler Skalierung.



Zeitverlauf-Anzeigefenstervor (oben) und nach der automatischen Umskalierung (unten)

Wir wollen nun - ausgehend vom bestehenden System - zu einem etwas komplexeren Beispiel übergehen. Unsere Regelstrecke soll zu einem geschlossenen Regelkreis ergänzt werden, der neben der Regelstrecke zunächst noch folgende Komponenten enthalten soll:

- einen P-Regler mit der Verstärkung $K_R = 2$

- ein Messglied in der Rückführung (PT_1-Glied mit $K = 1$, $T = 0.5$)
- einen Subtrahierer für den Soll-Ist-Vergleich

Das im folgenden beschriebene Beispiel befindet sich unter dem Namen DEMO2.BSY im Beispiel-Verzeichnis.

Da wir die bereits vorhandenen Blöcke auch weiterhin benötigen, reicht es aus, die bestehenden Verbindungen zu löschen. Dazu aktivieren wir zunächst die Regelstrecke (durch Einfachklick mit der linken Maustaste) und löschen dann über die entsprechende Schaltfläche der System-Toolbar die Ausgangsverbindung. Direkt im Anschluss daran können wir über die links daneben liegende Schaltfläche auch die Eingangsverbindung zum Generator löschen.

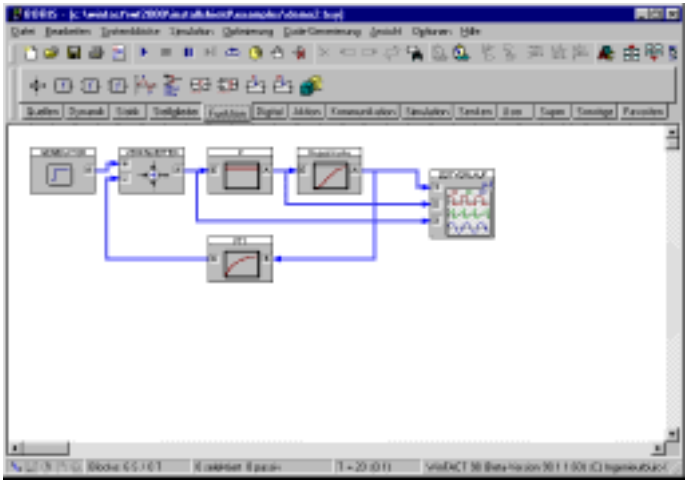
Die folgenden Schritte wollen wir nur noch grob beschreiben. Zunächst fügen wir folgende Blöcke ein:

den P-Regler (Palette *Dynamik*)

das PT_1-Messglied (Palette *Dynamik*)

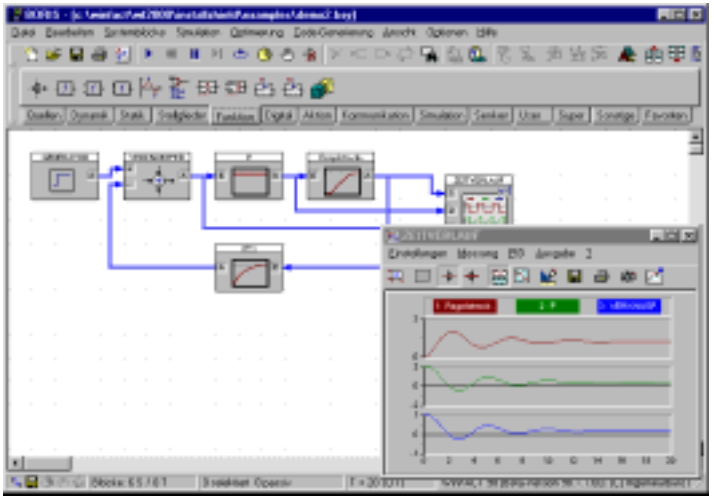
den Subtrahierer (Verknüpfer, Palette *Funktion*)

Im Anschluss daran ziehen wir die entsprechenden Verbindungen und parametrieren die einzelnen Blöcke. Beim Verknüpfer müssen wir insbesondere das negative Vorzeichen für die Rückführung vorsehen. Den Generator schalten wir auf die Betriebsart *Puls* um, so dass er einen Einheitssprung erzeugt. Da das Messglied in der Rückführung liegt, ist seine Orientierung etwas ungünstig. Dem können wir jedoch auf einfache Weise abhelfen, indem wir den Block um 180° drehen, so dass der Eingang rechts liegt. Die entsprechende Schaltfläche finden wir ebenfalls in der System-Toolbar. An den Zeitverlauf-Block schließen wir jetzt neben der Regelgröße, d. h. dem Ausgang der Regelstrecke, auch die Stellgröße (Ausgang des P-Reglers) und die Regelabweichung (Ausgang des Subtrahierers) an. Sofern wir alle Änderungen korrekt vorgenommen haben, sieht unser Anwendungsfenster nun in etwa wie folgt aus:



Kompletter Regelkreis

Um alle drei darzustellenden Signale in getrennten Diagrammen zu erfassen, deaktivieren wir im über den Menüpunkt EINSTELLUNGEN erreichbaren Dialog des Zeitverlaufs die Option *Alle Kurven in ein Diagramm* bzw. betätigen die entsprechende Toolbar-Schaltfläche. Die anschließende Simulation liefert folgende Ergebnisse:



Simulationsergebnisse

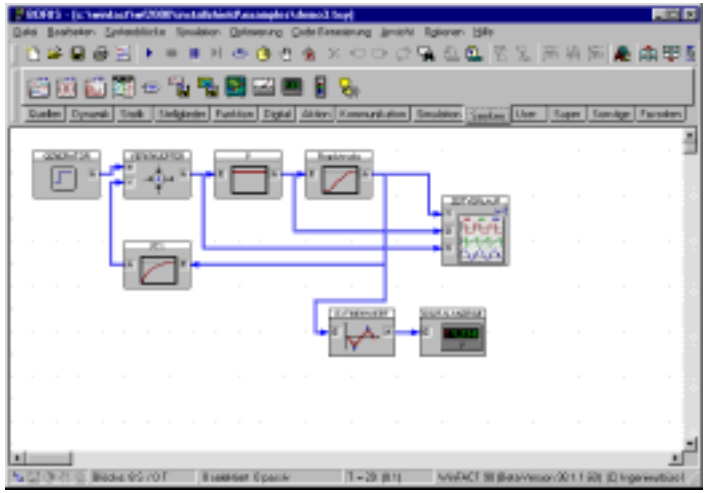
Wir wollen unser System ein letztes Mal erweitern, so dass wir einen genauen Wert für die maximale Ausgangsgröße erhalten. Dazu fügen wir zwei weitere Systemblöcke ein:

einen Extremwert-Block (Palette *Funktion*)

eine Digitalanzeige (Palette *Senken*)

Dieses Beispiel befindet sich unter dem Namen DEMO3.BSY im Beispielverzeichnis.

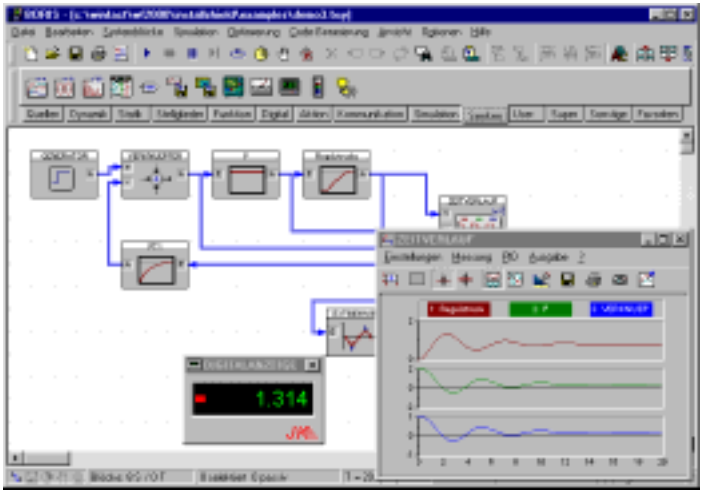
Unser Simulationssystem weist damit folgende Struktur auf:



System mit zusätzlicher Extremwertbestimmung

Vor der Simulation holen wir nun beide Anzeigefenster (Zeitverlauf und Digitalanzeige) aus der Symboldarstellung zurück. Nachfolgende Bildschirmgrafik zeigt das auf diese Weise erhaltene Simulationsergebnis. Dem Digitalinstrument können wir eine maximale Ausgangsgröße von ungefähr 1.31 entnehmen.

Wir wollen die einführenden Beispiele damit abschließen. Die folgenden Kapitel enthalten eine vollständige, detaillierte Beschreibung aller Leistungsmerkmale, Bedienoptionen und zur Verfügung stehenden Systemblöcke.



Simulationsergebnis

Aufbau der Simulationsstruktur

Komponenten des BORIS - Hauptfensters

Nachfolgende Grafik zeigt das BORIS-Hauptfenster mit seinen einzelnen Komponenten. Das Fenster enthält neben den Windows - Standardkomponenten die folgenden Bestandteile:

- Eine erste horizontale Toolbar (System-Toolbar) unterhalb des Menüs. Diese Toolbar enthält Schaltflächen für die am häufigsten benutzten Befehle. Über OPTIONEN | TOOLBARS | SYSTEMTOOLBAR SICHTBAR kann die Toolbar wechselweise zu- oder abgeschaltet werden.

- Eine zweite horizontale Toolbar (Optionen-Toolbar). Sie ermöglicht die schnelle Suche nach Text- oder Systemblöcken innerhalb der aktuellen Systemstruktur sowie die Änderung von Blockgrößen. Über OPTIONEN | TOOLBARS | OPTIONENTOOLBAR SICHTBAR kann die Toolbar wechselweise zu- oder abgeschaltet werden.
- Eine dritte, in mehrere Paletten aufgeteilte horizontale Toolbar (Systemblock-Toolbar). Sie ermöglicht den direkten Zugriff auf alle Systemblöcke. Die Palette *Favoriten* kann vom Anwender beliebig konfiguriert werden (siehe Abschnitt *Konfigurierung der Systemblock-Toolbar*). Über OPTIONEN | TOOLBARS | SYSTEMBLOCKTOOLBAR SICHTBAR kann die Toolbar wechselweise zu- oder abgeschaltet werden.
- Eine vertikale Toolbar am rechten Fensterrand (Farb-Toolbar). Über diese ist ein schnelles Ändern der Farbe von Verbindungen oder Kommentartexten möglich.
- Eine Statuszeile am unteren Fensterrand. Sie gibt die Anzahl der aktuell vorhandenen Systemblöcke[1], die Anzahl der selektierten bzw.

[1] In Klammern wird zusätzlich die Zahl der für Versionen mit eingeschränkter Blockzahl relevante Blockanzahl angegeben; hierbei werden Signalquellen, Signalsenken und Label-Blöcke nicht mitgezählt.

passiv gesetzten Blöcke[1] sowie die aktuellen Simulationsparameter in der Form T = T_{Simu} (ΔT) an. Dabei ist T_{Simu} die Simulationsdauer und ΔT die Simulationsschrittweite. Das nachfolgende Kürzel (oben *RK*) kennzeichnet das aktive Integrationsverfahren. Wurde die Betriebsart *Echtzeitsimulation* aktiviert, wird im Anschluss daran zusätzlich das Kürzel *RT* (für *Real Time*) angezeigt.

Während der Simulation zeigt die Statuszeile den Fortlauf der Simulation an, sofern diese Option nicht deaktiviert wurde. Am linken Rand der Statuszeile zeigen fünf Icons den aktuellen Systemzustand an:

 Autorouter aktiv

 System wurde seit der letzten Änderung noch nicht gespeichert

 Simulation läuft

 Breakpoint aktiv

 Optimierung läuft

- Das eigentliche Zeichenfenster zur Anzeige der Systemstruktur. Es ist über seine Bildlaufleisten sowohl in vertikaler als auch in horizontaler Richtung scrollbar; bei Mäusen mit Mausrad ist ein vertikales Scrollen auch über das Mausrad möglich. Zu Beginn befindet sich der sichtbare Ausschnitt in der linken oberen Ecke. Dieser Ausgangszustand kann jederzeit über OPTIONEN | BILDAUSSCHNITT IN URSPRUNG wiederhergestellt werden. Das Zeichenfenster weist standardmäßig ein Punktraster auf, an dem alle Systemblöcke mit der linken oberen Ecke ausgerichtet werden. Über die Menüfolge OPTIONEN | AN RASTER AUSRICHTEN lässt sich das automatische Ausrichten deaktivieren; die Systemblöcke sind dann beliebig plazierbar. Die Anzeige des Rasters selbst lässt sich über OPTIONEN | RASTER ANZEIGEN ausschalten.

Häufig möchte man – z. B. während der Simulation – die eigentlich Systemstruktur ausblenden. Dies ist über die Menüoption ANSICHT | AUF TOOLBAR VERKLEINERN/WIEDERHERSTELLEN bzw. die Schaltfläche  möglich. Das Hauptfenster von BORIS wird dann auf die Höhe der System-Toolbar verkleinert (siehe nachfolgende Bildschirmgrafik) bzw. seine ursprüngliche Höhe wiederhergestellt.

[1] Man beachte, dass bei der Anzeige der Systemblockanzahl nur "echte" Systemblöcke (also keine Textblöcke) gezählt werden, während bei der Anzahl der selektierten bzw. passiven Blöcke alle Blocktypen berücksichtigt werden!

Auf Toolbar verkleinertes BORIS-Hauptfenster

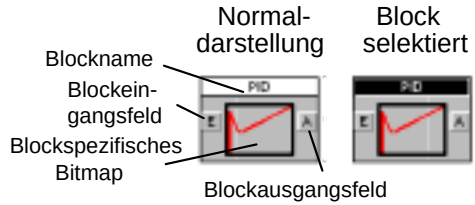
Einfügen und Bearbeiten von Systemblöcken

Das Einfügen und Bearbeiten von Systemblöcken umfasst folgende Möglichkeiten:

- *Einfügen* neuer Blöcke aus der Systemblock-Bibliothek
- *Selektieren* einzelner Blöcke oder Blockgruppen
- *Verschieben* einzelner Blöcke oder Blockgruppen
- *Löschen* einzelner Blöcke oder Blockgruppen
- *Drehen* einzelner Blöcke
- *Kopieren* und *Einfügen* einzelner Blöcke oder Blockgruppen
- *Deaktivieren* (Passivsetzen) einzelner Blöcke oder Blockgruppen
- *Parametrieren* einzelner Blöcke
- Ändern der *Blockgröße*

Wie ein Block aufgebaut ist

Folgende Grafik zeigt den Aufbau eines Systemblocks:



Er besteht aus folgenden Komponenten:

- Dem *Titelbalken*, der den Blocknamen enthält. Dieser entspricht per Voreinstellung dem Namen des Blocktypen, kann aber vom Anwender über den Parameterdialog modifiziert werden (max. 25 Zeichen). Im selektierten Zustand wird der Titelbalken invertiert. Ist der Blockname

zu lang, um vollständig angezeigt zu werden, wird er automatisch abgekürzt.

- Den *Ein-* und *Ausgangsgrößenfeldern*, die zum Ziehen von Verbindungen benötigt werden. Bei Blöcken mit nur einem Ein- und Ausgang ist das Eingangsfeld mit "E" und das Ausgangsfeld mit "A" gekennzeichnet; bei mehreren Ein- oder Ausgängen sind diese in der Regel durchnumeriert oder tragen spezielle Bezeichnungen (z. B. "R" und "S" beim RS-Flip-Flop).
- Einem *blockspezifischen Bitmap*, das den jeweiligen Blocktyp unmittelbar erkennen lässt.

Ein Block kann je nach Blocktyp bis zu fünfzig Ein- und Ausgänge besitzen.

Um einen neuen Block einzufügen...

...gibt es drei verschiedene Möglichkeiten:

1. Sie klicken mit der linken Maustaste die entsprechende Schaltfläche in der Systemblock-Toolbar an. Der Block wird dann automatisch an einer freien Stelle auf dem Arbeitsblatt plaziert.
2. Sie wählen den entsprechenden Blocktyp aus dem SYSTEMBLÖCKE-Untermenü des Hauptmenüs. Auch in diesem Fall wird der Block automatisch plaziert.
3. Sie klicken mit der linken Maustaste die entsprechende Schaltfläche in der Systemblock-Toolbar an, halten die Maustaste gedrückt und ziehen den Block per Drag & Drop an den gewünschten Ort. Beim Loslassen der Maustaste wird der Block dann dort plaziert.

Letztere Vorgehensweise empfiehlt sich vor allem dann, wenn bereits eine Vielzahl von Blöcken auf dem Arbeitsblatt plaziert wurde.

Wie Sie Blöcke selektieren

Zur Selektion einzelner Blöcke oder ganzer Blockgruppen gibt es verschiedene Möglichkeiten:

- ➢ Einen einzelnen Block selektieren Sie durch einfaches Anklicken mit der Maus. War zuvor ein anderer Block selektiert, so wird dessen Selektion zurückgenommen.
- ➢ Sollen zusätzlich weitere Blöcke selektiert werden, so erreichen Sie dies, indem Sie diese bei festgehaltener <Shift>- oder <Strg>-Taste anklicken.

Ein nochmaliges Anklicken eines bereits selektierten Blocks nimmt die Selektion wieder zurück.

- Alternativ dazu können Sie ganze Blockgruppen durch Aufziehen eines Rechtecks mit festgehaltener linker Maustaste selektieren. Alle Blöcke, die vollständig im Rechteck liegen, werden dabei selektiert.
- Über die Menüoption BEARBEITEN | ALLE BLÖCKE SELEKTIEREN lassen sich alle Blöcke gleichzeitig selektieren.

Das Anklicken einer beliebigen freien Stelle innerhalb des Zeichenbereichs nimmt alle Selektionen zurück.

So verschieben Sie Blöcke

Zum Verschieben eines Blocks oder mehrerer Blöcke gehen Sie wie folgt vor:

1. Selektieren Sie zunächst den zu verschiebenden Block bzw. die zu verschiebenden Blöcke.
2. Klicken Sie mit der linken Maustaste den Innenbereich des zu verschiebenden Blocks bzw. - bei der Verschiebung einer Blockgruppe - den Innenbereich eines beliebigen selektierten Blocks bei festgehaltener Maustaste und verschieben Sie die Blöcke in gewünschter Weise. Der Cursor wechselt dabei zur Kreuzform. Bei Erreichen des Fensterrands erfolgt automatisches Scrollen.

Nach der Verschiebung werden die Blöcke gegebenenfalls so ausgerichtet, dass sie sich nicht überlappen.

Verschieben des Gesamtsystems

Während der Systemkonfigurierung kann speziell bei komplexeren Systemen der Fall auftreten, dass links oder oberhalb des Systems weitere Systemblöcke eingefügt werden sollen, dort aber kein Platz mehr zur Verfügung steht. Daher bietet BORIS die Möglichkeit, das Gesamtsystem (d. h. alle Blöcke und Verbindungen) auf einfache Weise zu verschieben, ohne dass diese zuvor selektiert werden müssen. Die Verschiebung kann in alle Richtungen vorgenommen werden und erfolgt bei jedem Schritt um eine Rastereinheit. Die entsprechenden Menübefehle lauten:

OPTIONEN | SYSTEM NACH RECHTS VERSCHIEBEN

OPTIONEN | SYSTEM NACH LINKS VERSCHIEBEN

OPTIONEN | SYSTEM NACH UNTEN VERSCHIEBEN

OPTIONEN | SYSTEM NACH OBEN VERSCHIEBEN

Löschen von Blöcken

Das Löschen von Blöcken kann auf drei Arten erfolgen:

- Durch Selektieren der zu löschenden Blöcke und Wahl der Menüoption BEARBEITEN | BLOCK LÖSCHEN
- Durch Selektieren der zu löschenden Blöcke und Betätigung der Schaltfläche  der System-Toolbar
- Durch Selektieren der zu löschenden Blöcke und einen rechten Mausklick. Im daraufhin erscheinenden Popup-Menü wählen Sie die Option LÖSCHEN. Soll nur ein einzelner Block gelöscht werden, kann dieser direkt mit der rechten Maustaste angeklickt werden, ohne dass er zuvor selektiert werden muss.

Eingangs- und Ausgangsverbindungen der gelöschten Blöcke werden automatisch mitgelöscht.

So lassen sich Blöcke drehen

Die Orientierung eines Blocks kann um 180° gedreht werden, so dass die Eingänge auf der rechten Seite liegen (beispielsweise für Blöcke in einer Rückführung). Dazu wird der Block zunächst selektiert und dann über BEARBEITEN | BLOCK DREHEN oder Betätigung der Schaltfläche  gedreht. Alternativ dazu können Sie den Block mit der rechten Maustaste anklicken und im daraufhin erscheinenden Popup-Menü die Option DREHEN wählen. Die Blockverbindungen werden automatisch nachgeführt (dies gilt allerdings nicht für manuelle Verbindungen; siehe dazu später!).

Kopieren und Einfügen

Das Kopieren und Einfügen einzelner Blöcke oder ganzer Blockgruppen läuft über eine temporäre Datei mit Namen BOCOPY.TMP ab. Dabei werden die selektierten Blöcke, ihre Parameter und die Verbindungen der Blöcke untereinander kopiert bzw. eingefügt. Gehen Sie wie folgt vor:

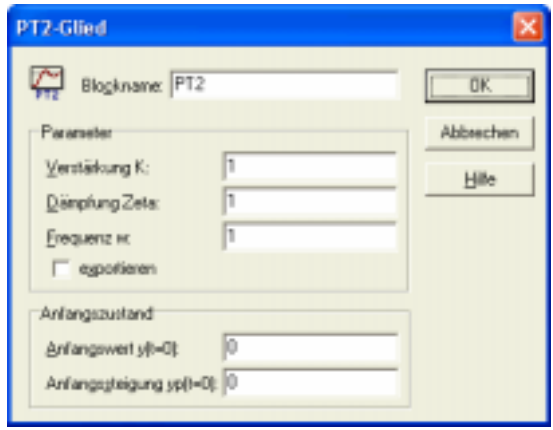
- Selektieren Sie die zu kopierenden Blöcke.
- Wählen Sie die Menüoption BEARBEITEN | KOPIEREN bzw. betätigen die Schaltfläche  der System-Toolbar.

➢ Zum Einfügen wählen Sie BEARBEITEN | EINFÜGEN bzw. die Schaltfläche . Der Cursor wechselt daraufhin seine Gestalt und Sie können die gewünschte Zielposition (linke obere Ecke) des einzufügenden Teilsystems durch Anklicken des Arbeitsblattes vorgeben.

Wie Sie Blöcken ihre Parameter verpassen

Die Parametrierung eines Systemblocks erfolgt am einfachsten durch einen Doppelklick mit der linken Maustaste innerhalb des Blocks. Alternativ dazu kann nach Selektion des Blocks die Menüfolge BEARBEITEN | BLOCK BEARBEITEN aufgerufen werden. Es erscheint dann der blockspezifische Parameterdialog, über den die gewünschten Parameteränderungen vorgenommen werden können. Folgende Grafik zeigt beispielhaft den Parameterdialog für ein schwingfähiges PT_2-Glied.

Der Parameterdialog weist - unabhängig vom Blocktyp - am oberen Rand ein Texteingabefeld für den Blocknamen auf, der in der Blockdarstellung jeweils im Titelbalken erscheint. Dieser Blockname entspricht in der Voreinstellung dem Namen des Blocktyps, kann vom Anwender aber beliebig geändert werden. Die Länge des Blocknamens darf jedoch 25 Zeichen nicht überschreiten. Darüber hinaus weist jeder Parameterdialog eine *Hilfe*-Schaltfläche auf, über die Informationen zum jeweiligen Blocktyp angefordert werden können.



Parameterdialog für schwingfähiges PT_2-Glied

Sein oder nicht sein? So setzen Sie Blöcke passiv

Häufig möchte man gewisse Teile der Simulationsstruktur kurzzeitig "ausschalten", z. B. um verschiedene Modellansätze vergleichen zu können, um File-Output-Blöcke zu deaktivieren oder um bei sehr komplexen Strukturen auch einfach nur Rechenzeit zu sparen. Statt diese Komponenten mühsam aus der Struktur entfernen und später wieder einfügen zu müssen, bietet Ihnen BORIS die Möglichkeit, Blöcke auf Mausklick *passiv zu setzen*. Passiv gesetzte Blöcke werden während der Simulation so behandelt, als wären sie nicht vorhanden. Blockeingänge, die von einem passiv gesetzten Block gespeist werden, werden also wie offene Eingänge behandelt (ein passiver Block in einer Reihenschaltung "unterbricht" die Verbindung also beispielsweise!).

Um einen einzelnen Block passiv zu setzen,

- klicken Sie den Block mit der *rechten* Maustaste an und wählen im daraufhin eingeblendeten Popup-Menü die Option PASSIV

oder (und das ist der umständliche Weg)

- selektieren Sie den Block und wählen im Hauptmenü die Menüoption BEARBEITEN | BLOCK PASSIV.

Wollen Sie mehrere Blöcke gleichzeitig passiv setzen, so

- selektieren Sie die Blöcke,
- klicken mit der rechten Maustaste auf das Arbeitsblatt und wählen im daraufhin eingeblendeten Popup-Menü die Option PASSIV

oder (wiederum umständlicher)

- selektieren Sie die Blöcke und wählen im Hauptmenü die Menüoption BEARBEITEN | BLOCK PASSIV.

Um einen passiven Block wieder zu aktivieren,

- klicken Sie den Block mit der *rechten* Maustaste an und wählen im daraufhin eingeblendeten Popup-Menü die Option AKTIV

oder (und das ist der umständliche Weg)

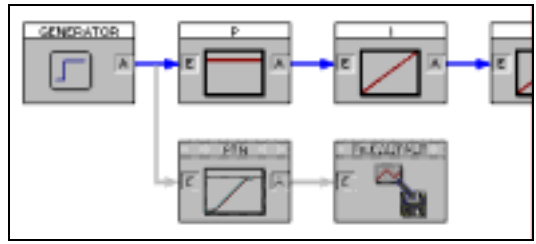
- selektieren Sie den Block und wählen im Hauptmenü die Menüoption BEARBEITEN | BLOCK AKTIV.

Um mehrere passive Blöcke zu aktivieren, verfahren Sie sinngemäß.

Um *alle* Blöcke zu aktivieren,

- wählen Sie die Menüoption BEARBEITEN | ALLE BLÖCKE AKTIVIEREN.

Passiv gesetzte Blöcke erkennen Sie am grauschattierten Block-Bitmap. Ebenso werden Verbindungen, die von passiven Blöcken ausgehen oder solche speisen, grau dargestellt, sofern diese Option nicht über den Anzeige-Dialog (OPTIONEN | ANPASSEN...) deaktiviert wurde.

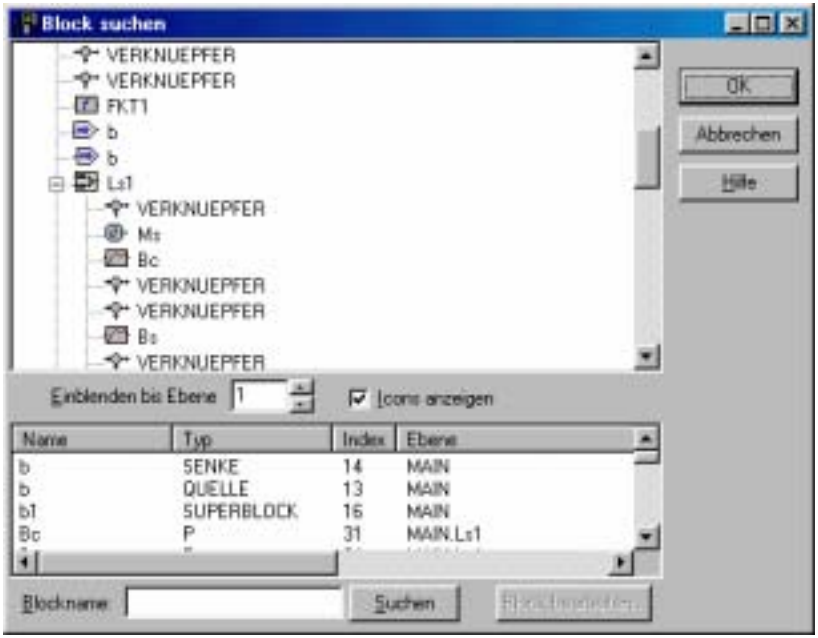


Passivsetzen von Blöcken: Der untere vom Generator ausgehende Zweig wurde hier deaktiviert.

Wo ist er denn? Blöcke suchen (und finden)

Beim Bearbeiten komplexer Simulationsstrukturen ist oft ein schneller Zugriff auf einen bestimmten Block notwendig, z. B. um seine Parameter zu ändern. Das Durchsuchen des Systems über die Bildlaufleisten ist hier oft sehr mühsam; außerdem wird der gesuchte Block in vielen Fällen gar nicht im aktuell geladenen System, sondern z. B. in einem der definierten Superblöcke (u. U. sogar in einem Superblock innerhalb eines Superblocks...) sein. BORIS bietet daher eine leistungsfähige Suchfunktion, die über BEARBEITEN | BLOCK SUCHEN... bzw. die Schaltfläche der System-Toolbar aufgerufen wird.

Der Suchdialog listet nach dem Aufruf im oberen Teil, der Baumstrukturansicht, zunächst alle Blöcke bis zur untersten Hierarchieebene auf. Über das Eingabefeld *Einblenden bis Ebene* können die Ebenen schrittweise ausgeblendet werden. Die untere Ansicht (Listenstruktur) enthält grundsätzlich alle Ebenen. Dabei wird die oberste Ebene (d. h. die aktuell geladene System- oder Superblockdatei) mit *MAIN* bezeichnet, untergeordnete Ebenen werden jeweils durch einen Punkt voneinander getrennt. So bezeichnet z. B. *MAIN.TASTERA* einen Block, der sich in der aktuellen Systemdatei im Superblock *TASTERA* befindet.



Der Block suchen-*Dialog von BORIS*

Alle Einträge der Listenstruktur sind alphabetisch bezüglich des Blocknamens geordnet. Durch Anklicken einer der Listenüberschriften kann die Sortierung jedoch auch nach einem anderen Kriterium (z. B. dem Blocktyp) erfolgen. Zum Auffinden eines gesuchten Blocks gibt es zwei Möglichkeiten: einen Doppelklick auf die entsprechende Baum- oder Listenansicht oder die Eingabe des Blocknamens (oder eines Teils davon, z. B. *GEN*) im Eingabefeld ***Blockname*** und Betätigung der ***Suchen***-Schaltfläche. Das weitere Verhalten von BORIS hängt nun vom gewählten Blocktyp ab:

- Ein Block *der obersten Hierachieebene* wird automatisch selektiert und der Bildausschnitt - soweit möglich - so verschoben, dass der gesuchte Block genau in der Fenstermitte plaziert ist.
- Wurde ein Block gewählt, der sich *in einem Superblock* befindet (gleichgültig in welcher Hierarchieebene), so wird der entsprechende Superblock automatisch zur Bearbeitung geladen. Der gesuchte Block wird in diesem Fall allerdings *nicht* automatisch mittig plaziert, so dass ggf. innerhalb der geladenen Superblock-Datei erneut die Suchfunktion aufgerufen werden muss!

Blöcke der obersten Hierarchieebene (mit Ausnahme von Superblöcken) können außerdem direkt aus dem Dialog heraus über die Schaltfläche *Block bearbeiten* bearbeitet werden.

Blöcke innerhalb der aktuellen Systemstruktur können alternativ auch direkt über die Optionen-Toolbar gesucht werden (siehe nachfolgende Bildschirmgrafik). Dazu wird der Name des gesuchten Blocks oder der Anfang des Blocknamens (hier *Motor*) in das Editierfeld eingegeben und der Suchvorgang über die *Go*-Schaltfläche gestartet. Wird ein passender Block gefunden, erscheint er in der linken oberen Ecke der Arbeitsfläche; andernfalls erfolgt eine Warnmeldung. Durch wiederholte Betätigung der *Go*-Schaltfläche kann nach weiteren passenden Blöcken gesucht werden.

Suchen von Blöcken über die Optionen-Toolbar

Ändern der Blockgröße

BORIS kann alle Systemblöcke in unterschiedlichen Größenstufen darstellen. Ein neu eingefügter Block wird in der Regel in der größten Stufe (100%) dargestellt. Diese Standard-Einstellung kann aber über den Anzeige-Dialog (erreichbar über OPTIONEN | ANPASSEN..., Eingabefeld *Standard-Blockgröße*) geändert werden.

Um die Blockgröße einzelner oder mehrerer Blöcke zu ändern, gehen Sie wie folgt vor:

1. Öffnen Sie das Blockgrößen-Fenster über ANSICHT | BLOCKGRÖßEN-FENSTER.
2. Selektieren Sie die zu modifizierenden Blöcke.
3. Geben Sie im Eingabefeld *Relative Größe* des Blockgrößen-Fensters die gewünschte Größe an und betätigen Sie die *Zuweisen*-Schaltfläche.

Das Blockgrößen-Fenster

Verbinden der Systemblöcke

So verbinden Sie zwei Blöcke miteinander

Alle Verbindungen zwischen Blöcken werden mausgesteuert gezogen. Da ein integrierter Autorouter automatisch für rechtwinklige, möglichst kreuzungsfreie Verbindungen sorgt, müssen in der Regel lediglich die beiden zu verbindenden Blöcke angewählt werden. Bei Bedarf kann der Autorouter über OPTIONEN | AUTOROUTER auch deaktiviert werden, so dass die Verbindungen auch manuell gelegt werden können. Der aktuelle Status wird in der Statuszeile grafisch angezeigt.

Um eine automatische Verbindung zu ziehen, gehen Sie wie folgt vor:

1. Zunächst ist das Ausgangsfeld des Blocks, von dem die gewünschte Verbindung ausgehen soll, mit der linken Maustaste anzuklicken. Der Mauszeiger wechselt dadurch seine Form und stellt nunmehr einen stilisierten Lötkolben dar. Ein *A* neben dem Cursor weist auf den aktivierten Autorouter hin.

2. Jetzt kann das Eingangsfeld des Zielblocks angewählt und durch einen Mausklick mit der linken Taste bestätigt werden. Sobald Sie sich über einem zulässigen Eingangsfeld befinden, wechselt der Cursor seine Farbe auf schwarz und es erscheint zusätzlich ein stilisiertes Fadenkreuz. Während der Mausbewegung wird vom Ausgangsblock ein "Gummiband" nachgeführt. Bei Erreichen des Zeichenfensterrands erfolgt ein automatisches Scrollen. Soll eine begonnene Verbindung rückgängig gemacht werden, so erreicht man dies durch Anklicken einer beliebigen freien Stelle innerhalb des Zeichenfensters.

Ziehen einer automatischen Verbindung

Nach regulärer Beendigung der Verbindung wird diese automatisch mit entsprechender Bepfeilung eingezeichnet. Die Verbindungen werden so gelegt, dass möglichst keine anderen Systemblöcke geschnitten werden. Sollte dies dennoch einmal vorkommen, kann man in den meisten Fällen durch leichtes Verschieben einzelner Blöcke den gewünschten Zustand herstellen. Verbindungen, die von demselben Blockausgang stammen, werden vom Autorouter in der Regel automatisch zusammengefasst.

Manuelle Verbindungen werden komplett vom Anwender gezogen und können prinzipiell beliebigen Verlauf haben. Zum Ziehen manueller Verbindungen muss der Autorouter ausgeschaltet sein. Um eine manuelle Verbindung zu ziehen, gehen Sie wie folgt vor:

Ziehen einer manuellen Verbindung

1. Klicken Sie das Ausgangsfeld des Blocks, bei dem die Verbindung starten soll, mit der linken Maustaste an.
2. Um einen Knickpunkt anzulegen, betätigen Sie die linke Maustaste. Eine Verbindung kann bis zu 8 Knickpunkte besitzen. Sofern die Option *An Raster ausrichten* aktiviert ist, werden die einzelnen Teilstücke automatisch rechtwinklig ausgerichtet, sofern sie eine bestimmte Neigung nicht überschreiten.
3. Um eine manuelle Verbindung abzuschließen, klicken Sie das Eingangsfeld des Blocks an, in dem die Verbindung enden soll.

Eine begonnene Verbindung können Sie durch *Doppel*klick auf eine freie Stelle des Arbeitsblattes jederzeit zurücknehmen.

Bearbeiten einer manuellen Verbindung

Manuelle Verbindungen können jederzeit modifiziert werden. Dazu gehen Sie wie folgt vor:

1. Klicken Sie die Verbindung mit der linken Maustaste an. Die Knickpunkte werden daraufhin markiert.
2. Die einzelnen Knickpunkte lassen sich nun bei festgehaltener linker Maustaste beliebig verschieben. Wird dabei zusätzlich die <Shift>- oder <Strg>-Taste gedrückt, werden die verschobenen Punkte automatisch auf ein virtuelles 5-Pixel-Raster ausgerichtet.
3. Um einen neuen Knickpunkt einzufügen, klicken Sie die Stelle, an der der neue Punkt eingefügt werden soll, doppelt an.

Umwandeln von Verbindungen

Manuelle Verbindungen können jederzeit in automatische Verbindungen umgewandelt werden und umgekehrt. Um eine manuelle Verbindung in eine automatische Verbindung (bzw. umgekehrt) zu überführen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf das Blockeingangs- bzw. -ausgangsfeld, in das die gewünschte Verbindung läuft.
2. Wählen Sie im daraufhin erscheinenden Popup-Menü die Option VERBINDUNG MANUELL bzw. VERBINDUNG AUTO.

Wollen Sie alle Verbindungen eines Blocks gleichzeitig modifizieren, so erreichen Sie dies wie folgt:

1. Selektieren Sie den Block.
2. Wählen Sie im Hauptmenü die Option BEARBEITEN | VERBINDUNGEN | EINGANGSVERBINDUNGEN MANUELL bzw. entsprechende.

Vom Ausgang eines Blocks können beliebig viele Verbindungen ausgehen. Jeder Blockeingang kann jedoch nur eine Verbindung erhalten. Die Verbindungen können auf verschiedene Weise dargestellt werden. Die entsprechenden Möglichkeiten werden über OPTIONEN | ANPASSEN angeboten.

Wie Sie Verbindungen löschen

Um eine einzelne Verbindung zu löschen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf das Blockeingangs- bzw. -ausgangsfeld, in das die gewünschte Verbindung läuft.
2. Wählen Sie im daraufhin erscheinenden Popup-Menü die Option LÖSCHEN.

Wollen Sie alle Verbindungen eines Blocks gleichzeitig löschen, so erreichen Sie dies wie folgt:

1. Selektieren Sie den Block.
2. Wählen Sie im Hauptmenü die Option BEARBEITEN | VERBINDUNGEN | EINGANGSVERBINDUNGEN LÖSCHEN bzw. BEARBEITEN | VERBINDUNGEN | AUSGANGSVERBINDUNGEN LÖSCHEN oder betätigen Sie in der System-Toolbar die Schaltfläche  bzw.  .

Jetzt wird`s bunt: Mehrfarbige Verbindungen

Manchmal ist es aus Gründen der Übersichtlichkeit erwünscht, einzelnen Verbindungen unterschiedliche Farben zu geben. Daher erlaubt BORIS für jede Verbindung eine individuelle Farbe. Die Farbe für neu eingefügte Verbindungen ist standardmäßig blau. Diese Einstellung kann aber über den Anzeige-Dialog (erreichbar über OPTIONEN | ANPASSEN...) verändert werden. Um die Farbe bereits vorhandener Verbindungen zu ändern, gehen Sie wie folgt vor. Um die Farbe einer einzelnen Ein- bzw. Ausgangsverbindung eines Blocks zu ändern,

- klicken Sie mit der rechten Maustaste auf das Blockeingangs- bzw. -ausgangsfeld, in das die gewünschte Verbindung läuft und
- wählen im daraufhin erscheinenden Popup-Menü die Option VERBINDUNGSFARBE....

Um die Farbe aller Ein- bzw. Ausgangsverbindungen eines Blocks gleichzeitig zu ändern,

- selektieren Sie den Block und

➢ wählen im Hauptmenü die Option BEARBEITEN | VERBINDUNGEN | FARBE EINGANGSVERBINDUNGEN ... bzw. entsprechende.

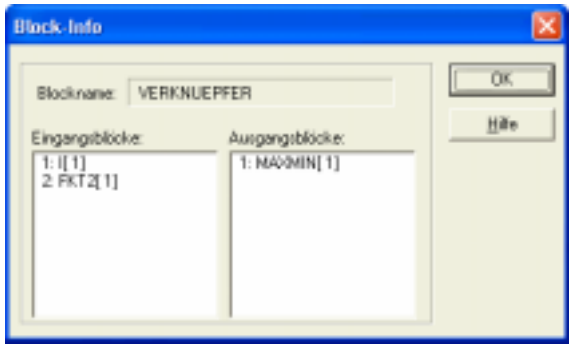
Zur Auswahl der gewünschten Farbe erscheint jeweils der entsprechende Windows-Farbauswahl-Standarddialog.

Alternativ dazu können Verbindungsfarben auch über die Farb-Toolbar modifiziert werden:

- Ist aktuell kein Block selektiert, ändert ein linker Mausklick auf die Toolbar die voreingestellte Farbe für neue Verbindungen (wird im obersten Feld der Toolbar angezeigt).
- Sind ein oder mehrere Blöcke selektiert, ändert ein linker Mausklick die Farbe aller Eingangsverbindungen, ein rechter Mausklick die Farbe aller Ausgangsverbindungen. Die voreingestellte Farbe für neue Verbindungen bleibt davon unberührt.

Anzeige der Blockverbindungen

Um sich bei komplexen Systemen einen Überblick über die Verbindungen der Blöcke untereinander zu verschaffen, kann die Menüfolge BEARBEITEN | BLOCK-INFO... herangezogen werden. Der entsprechende Dialog listet dann alle mit dem selektierten Block verbundenen Blöcke - getrennt nach Ein- und Ausgängen - auf.



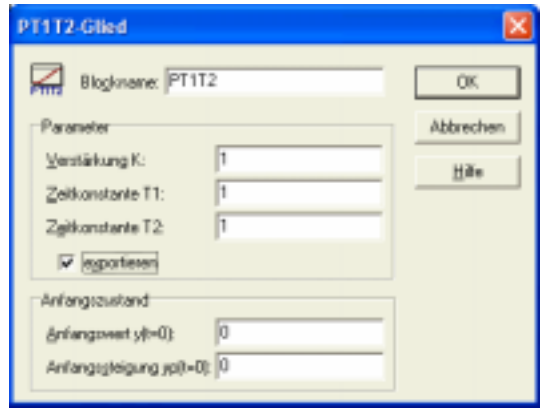
Block-Info-Dialog

Arbeiten mit Exportparametern

Eine Vielzahl von Systemblocktypen besitzt sogenannte *Exportparameter*. Dies sind Blockparameter, die dem Anwender einen "Zugriff von außen" gestatten, der für verschiedene Zwecke genutzt werden kann:

- Aus der aktuellen Systemebene heraus können Exportparameter über einen globalen Dialog gesetzt oder auch aus einer Datei eingelesen werden (s. u.).
- Innerhalb eines Superblocks können Exportparameter benutzt werden, um diese aus dem Superblock in die darüberliegende Ebene zu exportieren und von dort zu modifizieren. Einzelheiten dazu finden Sie im Abschnitt *Arbeiten mit Superblöcken*.
- Fließkomma-Exportparameter innerhalb der aktuellen Systemebene können durch eine numerische Parameteroptimierung optimiert werden. Näheres dazu finden Sie im Abschnitt *Numerische Optimierung von Systemparametern*.
- Fließkomma-Exportparameter können aus der Simulation heraus über PARMOD-Blöcke (Parameter-Modifizierer) modifiziert werden. Ihr aktueller Wert kann ferner über einen PARVAL-Block ausgelesen werden.
- Auf Basis von Fließkomma-Exportparametern können im so genannten *Batch-Betrieb* automatisch ganze Simulationsreihen durchgeführt werden. Hinweise dazu finden Sie im Abschnitt *Simulationen im Batch-Betrieb*.

Um einen exportfähigen Blockparameter als Exportparameter aktiv werden zu lassen, muss dieser zunächst "freigeschaltet" werden. Nachfolgende Grafik zeigt den Parameterdialog eines PT_1T_2-Glieds. Bei diesem Blocktyp können die Verstärkung K sowie die Zeitkonstanten T_1 und T_2 exportiert werden. Dazu ist das Auswahlfeld *exportieren* zu aktivieren. Es sind jeweils nur alle Exportparameter eines Blocks gleichzeitig aktivierbar bzw. deaktivierbar.



Parameterdialog eines PT1T2-Blocks mit aktivierten Exportparametern

Sollen alle verfügbaren Exportparameter der aktuellen Systemstruktur gleichzeitig aktiviert werden, so ist dies über BEARBEITEN | EXPORTPARAMETER | ALLE AKTIVIEREN möglich; über BEARBEITEN | EXPORTPARAMETER | ALLE DEAKTIVIEREN können alle Parameter wieder deaktiviert werden. Über die Menüoption BEARBEITEN | EXPORTPARAMETER | BEARBEITEN lassen sich alle aktivierten Exportparameter der aktuellen Systemstruktur modifizieren.

Die Zuordnung der Exportparameter wird dabei über den *Blocknamen* vorgenommen; alle Blöcke, die Parameter exportieren, sollten daher möglichst unterschiedliche Blocknamen aufweisen, um Fehlzuordnungen zu vermeiden. Über die Menüoption DATEI | AUF DOPPELTE BLOCKNAMEN ÜBERPRÜFEN... kann das System ggf. überprüft werden. In der Spalte *Typ* des Dialogs wird der jeweilige Parametertyp (*F* für Fließkomma, *I* für Ganzzahl und *S* für String) angezeigt. Über die Schaltfläche *Aus Datei...* können die Parameter alternativ aus einer Textdatei eingelesen werden. Diese muss in jeder Zeile einen Parameterwert enthalten; für obiges Beispiel also in der ersten Zeile den Parameterwert für Block *Zweipunkt*, Parameter *yMin*, in der zweiten Zeile den Wert für Block *Zweipunkt*, Parameter *yMax* usw.

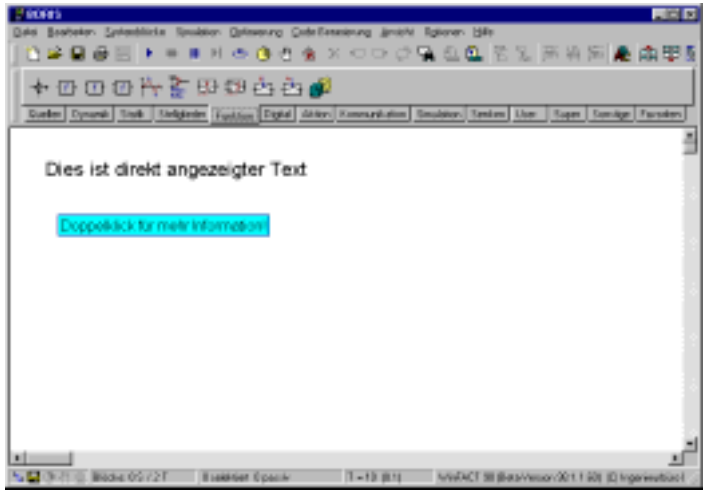
Dialog zur Bearbeitung von Exportparametern

Textblöcke, Bitmaps und Rahmen

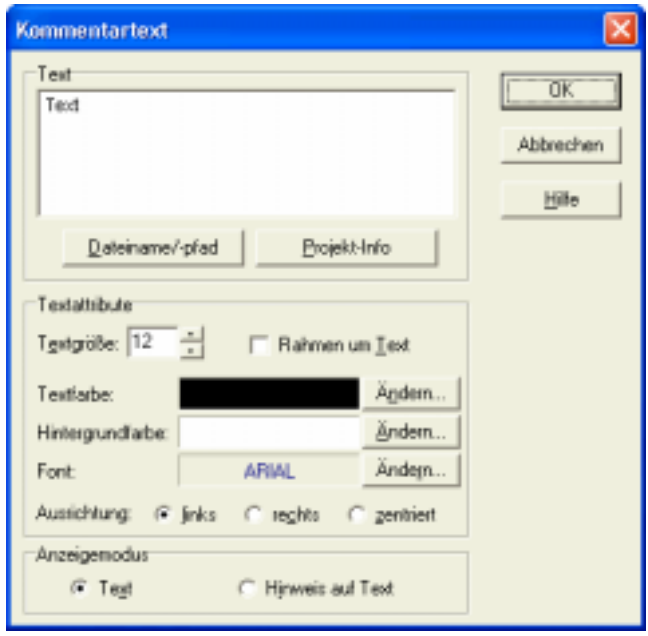
Arbeiten mit Textblöcken

Neben den eigentlichen Systemblöcken besteht in BORIS die Möglichkeit, beliebige Kommentartexte in das Strukturbild einzufügen. Diese Texte können in verschiedenen Farben und Schriftgrößen bzw. -typen gewählt und ebenso wie Systemblöcke verschoben und auch wieder gelöscht werden. Wahlweise kann ein Text dabei entweder direkt auf dem Arbeitsblatt erscheinen oder aber nur ein Hinweis auf den Text selbst, der dann erst durch einen Doppelklick auf diesen Hinweis erscheint. Letztere Möglichkeit ist insbesondere dazu geeignet, längere Informationstexte in einer Systemstruktur „unterzubringen“.

Zum Einfügen eines neuen Kommentartextes dient die Menüfolge BEARBEITEN | TEXT EINFÜGEN bzw. die Schaltfläche . Der Cursor wechselt daraufhin seine Gestalt und der voreingestellte Text *Text* kann per Drag & Drop auf dem Arbeitsblatt plaziert werden. Durch einen Doppelklick kann dieser anschließend modifiziert werden. Ein Verschieben des Textes ist wie bei Systemblökken mit festgehaltener linker Maustaste möglich.



Direkt angezeigter Text (oben) und Hinweis auf versteckten Text (unten)



Dialog zur Modifikation von Textblöcken

Die Text- bzw. Hintergrundfarbe kann alternativ auch über die Farb-Toolbar von BORIS geändert werden. Ein linker Mausklick ändert dabei die Textfarbe

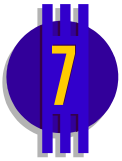
des aktuell selektierten Textblocks, ein rechter Mausklick die Hintergrundfarbe.

Suchen nach Texten

Textblöcke können auch benutzt werden, um bestimmte Teilstrukturen innerhalb der aktuellen Systemstruktur anhand von Beschriftungstexten auf schnelle Weise aufzufinden. Zu diesem Zweck dient die Menüoption BEARBEITEN | TEXT SUCHEN... (in diesem Fall werden alle vorhandenen Texte innerhalb der aktuellen Struktur aufgelistet) oder die Optionen-Toolbar (siehe nachfolgende Bildschirmgrafik). Dazu wird der zu suchende Text oder seine ersten Buchstaben (hier *abc*) in das Editierfeld eingegeben und der Suchvorgang über die *Go*-Schaltfläche gestartet. Wird ein passender Textblock gefunden, erscheint dieser in der linken oberen Ecke der Arbeitsfläche; andernfalls erfolgt eine Warnmeldung.

Suchen von Textblöcken über die Optionen-Toolbar

Einfügen von Bitmaps

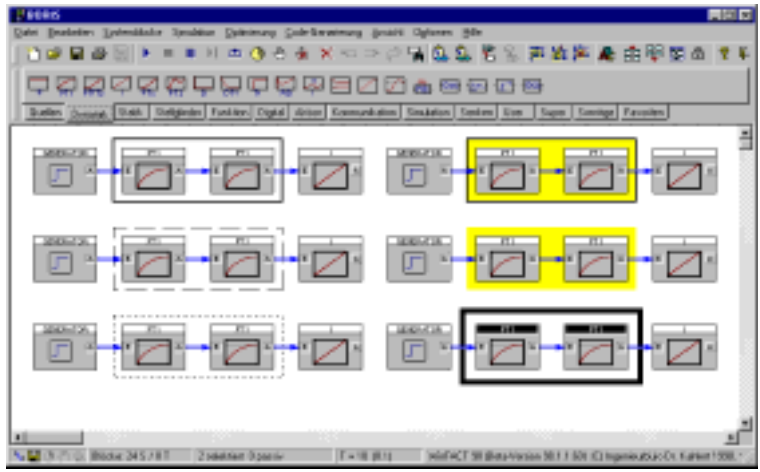


Neben Textinformationen können auch Bitmap-Grafiken in die Systemstruktur eingefügt werden; dazu dient die Menüoption BEARBEITEN | BITMAP-GRAFIK EINFÜGEN bzw. die Schaltfläche  . Die eingefügte Grafik kann auf Wunsch transparent dargestellt und in ihrer Größe vielfältig modifiziert werden (siehe nachfolgende Bildschirmgrafik). Man beachte dabei, dass gewöhnliche Systemblöcke und Textblöcke immer *vor*, Verbindungen aber immer *hinter* der Bitmap-Grafik erscheinen.

Dialog zur Modifikation von Bitmap-Grafiken

Die Rahmenfunktion von BORIS

Ein weiteres Feature zur besseren Dokumentation von Systemstrukturen stellt die Rahmenfunktion von BORIS dar. Sie ermöglicht es, zusammengehörige Blöcke durch einen umgebenden Rahmen optisch zusammenzufassen. Auf die Simulation hat diese Funktion natürlich keinerlei Einfluss.



Verschiedene Typen von Gruppenrahmen

Um einen Rahmen einzufügen,...

- selektieren Sie zunächst die einzurahmenden Blöcke,

➢ wählen Sie dann die Menüfolge BEARBEITEN | GRUPPENRAHMEN | RAHMEN EINFÜGEN bzw. betätigen die Schaltfläche der System-Toolbar.

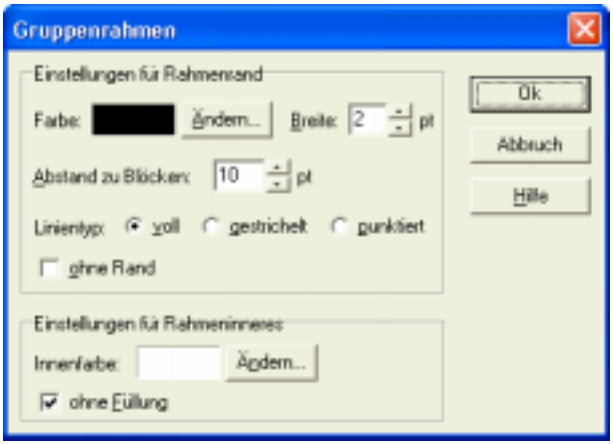
Anmerkung: Rahmen sind *statische* Gebilde, die in keinerlei direktem Bezug zu den eingerahmten Blöcken stehen. Sie können daher weder verschoben werden, noch werden sie beim Löschen von Blöcken automatisch mitgelöscht!

Erscheinungsbild des Rahmens

Der eingefügte Rahmen hat standardmäßig acht Pixel Abstand zum blockumgebenden Rechteck, einen schwarzen Rand mit Volllinie und ein Pixel Breite sowie keine Füllung. Diese Parameter lassen sich jedoch vom Anwender ändern. Dazu gehen Sie wie folgt vor:

1. Selektieren Sie einen beliebigen Block innerhalb des Rahmens.
2. Wählen Sie BEARBEITEN | GRUPPENRAHMEN | RAHMEN BEARBEITEN oder betätigen Sie die Schaltfläche der System-Toolbar.

Sie gelangen auf diese Weise in folgenden Dialog, aus dem heraus die Parameter modifizierbar sind.



Dialog zum Bearbeiten von Rahmen

Wie Sie Rahmen wieder entfernen können

Um einen Rahmen zu löschen, gehen Sie wie folgt vor:

1. Selektieren Sie einen beliebigen Block innerhalb des Rahmens.

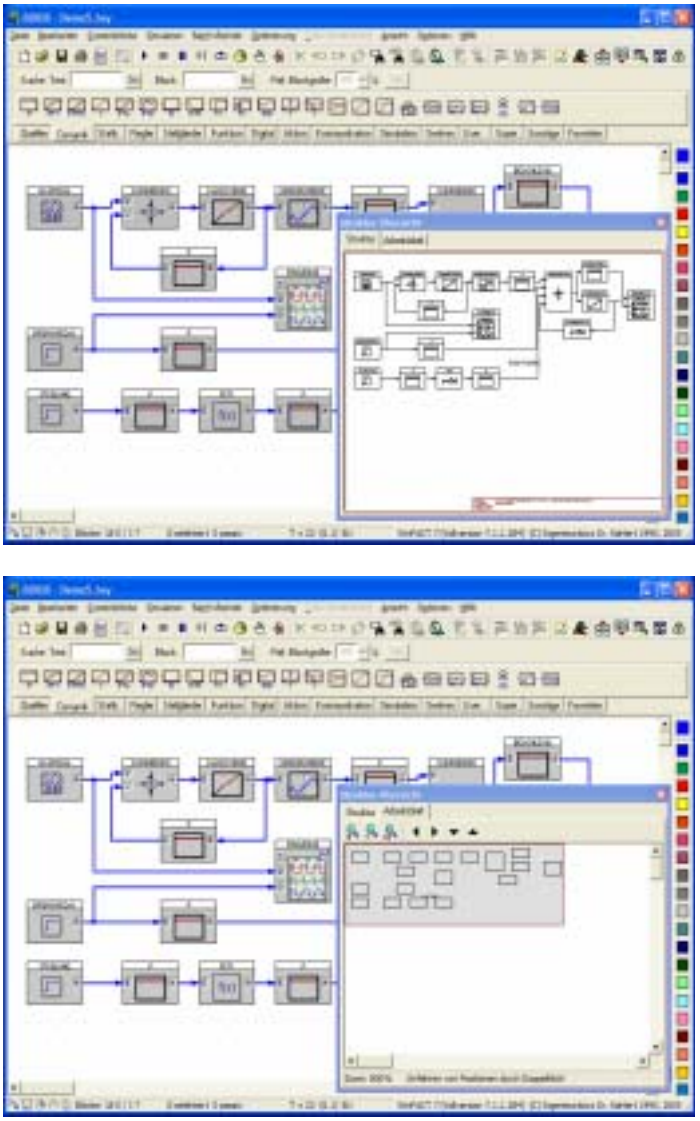
2. Wählen Sie BEARBEITEN | GRUPPENRAHMEN | RAHMEN LÖSCHEN oder betätigen Sie die Schaltfläche der System-Toolbar.

Struktur-Übersicht

Bei komplexeren Systemen reicht der dargestellte Bildausschnitt des Zeichenfensters in der Regel nicht zur Darstellung des Gesamtsystems aus. Um sich in solchen Fällen einen Überblick über die Gesamtstruktur zu verschaffen, besteht die Möglichkeit, ein zusätzliches Fenster mit einer maßstabsgetreuen, aber auf die Fenstergröße gezoomten Gesamtstruktur oder einem zoombaren Teilausschnitt zu öffnen. Dieses Fenster erhält man über ANSICHT | STRUKTUR-ÜBERSICHT bzw. die Schaltfläche der System-Toolbar. Das Fenster kann beliebig vergrößert und verkleinert werden und wird - solange es sichtbar ist - automatisch aktualisiert. Das Strukturübersicht-Fenster besitzt zwei mit *Struktur* bzw. *Arbeitsblatt* beschriftete Paletten, die unterschiedliche Sichtarten der Systemstruktur bieten (siehe auch nachfolgende Bildschirmgrafiken):

- In der Ansicht *Struktur* wird die gesamte Systemstruktur einschließlich der Verbindungen angezeigt, wobei die Systemblöcke durch s/w-Bitmaps dargestellt werden; diese Ansicht entspricht derjenigen, die beim Ausdrucken der Systemstruktur verwendet wird.
- In der Ansicht *Arbeitsblatt* wird ein zoom- und verschiebbarer Teilausschnitt der Gesamtstruktur dargestellt; der aktuelle Zoomfaktor wird in der Statuszeile des Fensters angezeigt. Wird die Maus über einen Block bewegt, werden Blocktyp, -name und –index in einem Hinweisfenster angezeigt. In dieser Ansicht wird aus Gründen der Übersichtlichkeit auf die Darstellung der Block-Bitmaps und der Verbindungen verzichtet.

In beiden Darstellungsformen wird durch einen Doppelklick mit der linken Maustaste innerhalb des Fensters der aktuelle Strukturausschnitt des BORIS-Hauptfensters so verschoben, dass die angeklickte Position in der Mitte des neuen Strukturausschnitts liegt. Auf diese Weise ist ein schnelles Anfahren bestimmter Positionen innerhalb der Systemstruktur möglich.



Struktur-Übersicht für ein einfaches System in beiden Darstellungsmodi

Datei-Operationen

Alle BORIS-Systemdateien sind ASCII-Dateien, die gegebenenfalls mit einem normalen Texteditor betrachtet oder "nachbearbeitet" werden können (letzterer Schritt ist nur in Ausnahmefällen anzuraten!). Es ist zu unterscheiden zwischen *regulären Systemdateien* (Extension BSY) und *Superblockdateien* (Extension SBL). Letztere sind Spezialfälle von Systemdateien und enthalten zu Beginn einen Dateiheader mit speziellen Informationen über die Struktur des Superblocks. Reguläre Systemdateien können in Superblockdateien umgewandelt werden - genauso wie umgekehrt. Auf das Arbeiten mit Superblöcken und ihren Dateien wird in einem separaten Abschnitt im Detail eingegangen. In diesem Abschnitt werden lediglich reguläre Systemdateien betrachtet.

Öffnen einer Datei

Eine existierende System- oder Superblockdatei können Sie wie folgt laden:

- Wählen Sie DATEI | DATEI ÖFFNEN... bzw. betätigen Sie die Schaltfläche der System-Toolbar und geben Sie den Dateinamen an.
- Die letzten fünf bearbeiteten Dateien erscheinen am unteren Ende des DATEI-Menüs und können von dort direkt geladen werden.

Der Dateiname der aktuellen Datei - sofern sie bereits einmal abgespeichert wurde - wird in der Titelzeile des BORIS - Hauptfensters angezeigt.

Um eine Datei zu speichern...

- ... wählen Sie die Menüfolge DATEI | DATEI SPEICHERN oder betätigen die Schaltfläche der System-Toolbar. Wollen Sie die Datei unter einem neuen Namen oder einem anderen Typ (z. B. eine Superblockdatei als reguläre Systemdatei oder umgekehrt) speichern, wählen Sie stattdessen DATEI | DATEI SPEICHERN UNTER...

Dateitypen

Anlegen einer neuen Datei

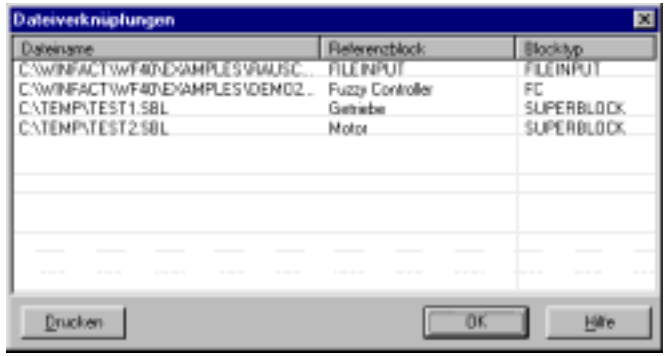
Über die Menüfolge DATEI | NEU oder die Schaltfläche wird BORIS in den Programmzustand beim Aufruf zurückversetzt. Sofern die aktuelle Datei seit der letzten Speicherung modifiziert wurde, erfolgt zuvor eine Sicherheitsabfrage.

Zusammenfügen von Dateien

Der aktuell geladenen Datei können weitere Dateien hinzugefügt werden. Zu diesem Zweck dient die Menüfolge DATEI | SYSTEMDATEI HINZUFÜGEN... mit nachfolgender Eingabe des Namens der hinzuzufügenden Systemdatei. Die zugeladene Systemstruktur wird unterhalb des aktuellen Systems angefügt. Das Hinzufügen beschränkt sich auf Blöcke, Verbindungen, Texte und Rahmen. Simulationsparameter, digitale Pegel usw. bleiben beim Hinzufügen unverändert.

Dateiverknüpfungen

Eine ganze Reihe von Systemblöcken (z. B. File-Inputs, Fuzzy Controller, Superblöcke, ...) benötigen ihrerseits Dateien für die Simulation. Soll daher beispielsweise eine Simulationsstruktur weitergegeben werden, so reicht nicht die Weitergabe der entsprechenden Systemdatei aus, sondern alle mit ihr verknüpften Dateien müssen ebenfalls mitkopiert werden. Die Menüfolge DATEI | DATEIVERKNÜPFUNGEN ... dient dabei als Hilfestellung, indem sie einen Dialog mit allen verknüpften Dateien auflistet. Angezeigt werden jeweils der Name der verknüpften Datei sowie Name und Typ des entsprechenden Systemblocks.



Anzeige von Dateiverknüpfungen

Steuerung der Simulation

Simulationsparameter

Bevor die Simulation gestartet werden kann, müssen in der Regel die Simulationsparameter gewählt werden. Der Dialog zur Wahl der Simulationsparameter wird über SIMULATION | PARAMETER... bzw. die Schaltfläche verfügbar und ist in drei Paletten aufgeteilt.

Grundeinstellungen

Palette Grundeinstellungen

Die einzelnen Optionen haben nachfolgende Bedeutungen:

Option	Bedeutung
Simulationsdauer	Die Simulationsdauer T_{Simu} gibt an, bis zu welchem Zeitpunkt die Simulation durchgeführt wird, sofern sie nicht vorher vom Anwender abgebrochen wird.
Schrittweite	Die Simulationsschrittweite ΔT gibt die Diskretisierungsschrittweite für die Simulation an und beeinflusst damit die Genauigkeit der erhaltenen Simulationsergebnisse. Wird ΔT zu groß gewählt, entstehen Diskretisierungsfehler, die im Extremfall zu einer völligen Verfälschung der Ergebnisse durch numerische Instabilität führen können. Als Anhaltspunkt für eine geeignete Wahl gilt, dass ΔT etwa 1/10 der kleinsten im System vorkommenden Zeitkonstanten nicht überschreiten sollte (s. auch *Integrationsverfahren*).
Echtzeit	Ist dieses Optionsfeld markiert, erfolgt die Simulation in Echtzeit, d. h. die unter *Schrittweite* eingestellte Simula-

tionsschrittweite entspricht der tatsächlichen Zeit zwischen zwei Simulationsschritten.

Fortschrittanzeige

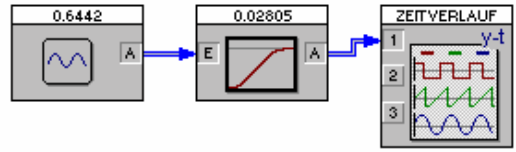
Diese Einstellung legt fest, auf welche Art der Fortlauf der Simulation angezeigt wird. In der Einstellung *grafisch* erfolgt die Anzeige in Form eines fortlaufenden Balkens in der Statuszeile, in der Einstellung *numerisch* durch Angabe der verstrichenen Zeit. Bei zeitkritischen Simulationen empfiehlt sich die Einstellung *keine*.

Kontrolle auf Bereichsüberschreitung

Ist diese Option aktiviert, so werden alle Zustandsgrößen des Systems bei jedem Simulationsschritt überprüft. Überschreitet eine der Größen betragsmäßig den Wert 10^{20}, so wird die Simulation mit einer entsprechenden Meldung abgebrochen. Da diese Überprüfung auf Bereichsüberschreitung einen erhöhten Rechenaufwand mit sich bringt, verläuft die Simulation bei aktiver Überprüfung geringfügig langsamer.

Kontrollausgabe in Blocktitel

Diese Option ist besonders nützlich bei der Suche nach Fehlern innerhalb der Systemstruktur. Ist sie aktiviert, erscheint während der Simulation im Blocktitel jedes Blocks (Ausnahme: Superblöcke) die aktuelle Ausgangsgröße des Blocks. Bei Blöcken mit mehreren Ausgängen kann allerdings nur der erste Ausgang angezeigt werden.

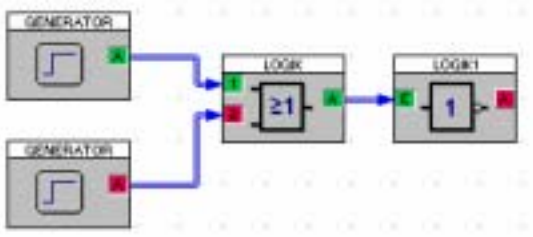


Kontrollausgabe in Blocktitel

Farbkennzeichnung der Signalpegel

Diese Option erleichtert insbesondere die Analyse von Systemen mit Digitalbausteinen. Ist sie aktiviert, so werden alle Blockausgänge sowie die mit Verbindungen belegten Blockeingänge farblich gekennzeichnet, je nachdem, ob der zugehörige aktuelle Signalwert LOW-Pegel (bzw. einen Wert unterhalb des Schaltpegels) oder HIGH-Pegel (bzw. einen Wert oberhalb des Schaltpegels) besitzen. LOW-Pegel wird durch eine rote Farbe, HIGH-Pegel durch eine grüne Farbe gekennzeichnet. Nicht verbundene Blockeingänge bleiben ohne Farb-

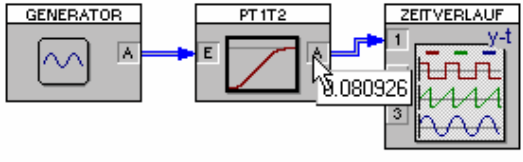
kennzeichnung.



Farbkennzeichnung von Signalpegeln

ToolTip-Ausgabe

Ist diese Option aktiviert, erscheint beim Überstreichen von Blockein- und -ausgängen mit der Maus ein kleines Fenster (*ToolTip*-Fenster), in dem der zugehörige Signalwert angezeigt wird.

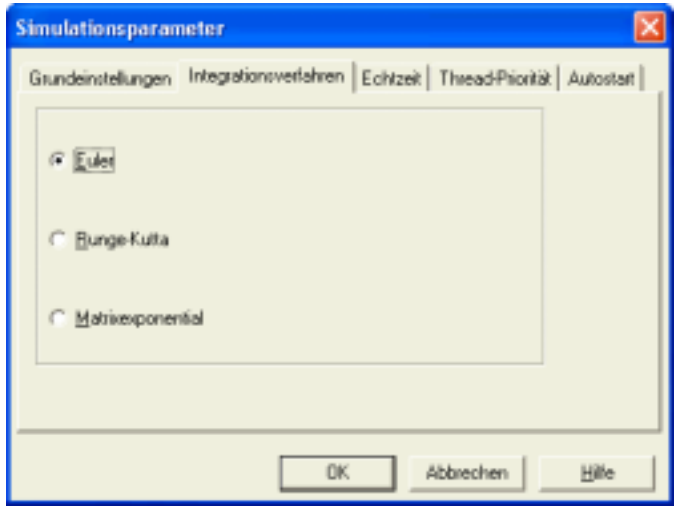


ToolTip-Ausgabe

Akustisches Signal bei Simulationsende

Über diese Option kann ein Signalton bei Beendigung bzw. Abbruch der Simulation erzeugt werden.

Integrationsverfahren



Palette Integrationsverfahren

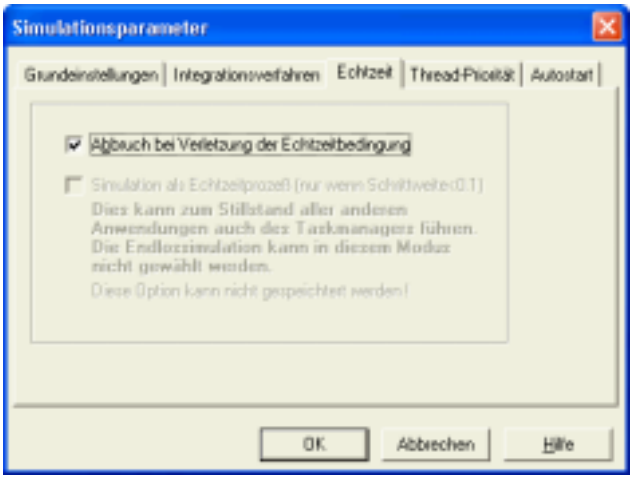
Diese Palettenseite erlaubt die Wahl des numerischen Integrationsverfahrens zur Simulation der dynamischen Systemkomponenten. In der aktuellen Version von BORIS sind das explizite Euler-Verfahren, das Runge-Kutta-Verfahren 4. Ordnung sowie das Matrizenexponentialverfahren verfügbar [3].

- Das Euler-Verfahren ist das einfachste aller Integrationsverfahren und weist damit den geringsten Rechenzeitaufwand auf, da pro Simulationsschritt nur ein Funktionswert berechnet werden muss. Es besitzt jedoch lediglich die Fehlerordnung $O(\Delta T^2)$ und eignet sich daher nur für die Simulation einfacher Systeme in Verbindung mit kleinen Simulationsschrittweiten. Speziell für stark schwingfähige Systeme oder Systeme mit stark unterschiedlichen Zeitkonstanten ("steife" Systeme) sollte das Verfahren in der Regel nicht eingesetzt werden.
- Das Runge-Kutta-Verfahren weist demgegenüber numerisch erheblich bessere Eigenschaften auf - es besitzt die lokale Fehlerordnung $O(\Delta T^5)$. Da bei diesem Verfahren jedoch für jeden Simulationsschritt vier Funktionsauswertungen erfolgen müssen, ist der Rechenzeitaufwand höher als beim Euler-Verfahren. Dies ist i. a. jedoch unerheblich. Bei sprungförmigen Änderungen von Eingangssignalen (Beispiel: Berechnung von Impulsantworten) kann es beim Einsatz dieses Verfahrens allerdings zu Ungenauigkeiten im Übergangsbereich kommen. In

diesen Fällen sollte daher besser das Euler-Verfahren in Verbindung mit einer kleineren Schrittweite eingesetzt werden.

- Das Matrizenexponentialverfahren eignet sich nur für lineare Systeme, besitzt dort jedoch eine im Prinzip unendlich hohe Fehlerordnung. Es sollte speziell dann eingesetzt werden, wenn z. B. Übertragungsfunktionen hoher Ordnung (> 3) als Systemblöcke auftreten. Wurde dieses Verfahren gewählt, so werden die nichtlinearen dynamischen Systemblöcke (nichtlineare Dgl.-Systeme) mit dem Runge-Kutta-Verfahren simuliert.

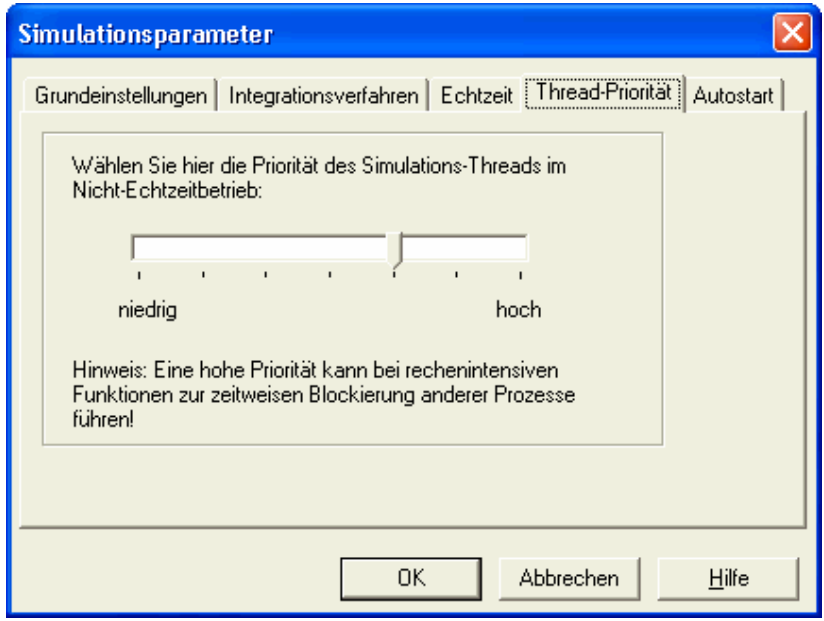
Echtzeitsimulation



Palette Echtzeitsimulation

Das Optionsfeld *Abbruch bei Verletzung der Echtzeitbedingung* bestimmt das Verhalten von BORIS bei Echtzeitsimulation für den Fall, dass die gewünschte Simulationsschrittweite aus irgendeinem Grund (z. B. wegen einer sehr komplexen Systemstruktur mit vielen rechenintensiven Blöcken) nicht eingehalten werden kann. Ist die Option aktiviert, bricht BORIS die Simulation in diesem Fall mit einer entsprechenden Warnmeldung ab.

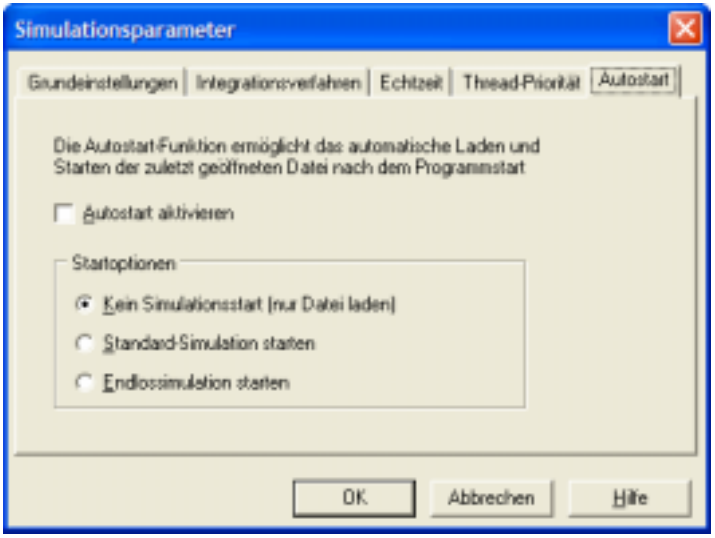
Priorität des Simulations-Threads



Palette Thread-Priorität

Simulation und Ausgabe laufen in BORIS in verschiedenen Programmfäden (Threads) ab. Für den Nicht-Echtzeitbetrieb kann über diese Palette die Priorität des Simulations-Threads eingestellt werden. Je höher diese gewählt wird, um so schneller läuft die eigentliche Simulation ab, um so seltener erfolgt allerdings eine Auffrischung der Ausgabe (z. B. des Simulationsfortschrittes in der Statuszeile). Um eine kontinuierliche Auffrischung zu erreichen, kann die Priorität des Simulations-Threads ggf. auf Kosten einer niedrigeren Simulationsgeschwindigkeit herabgesetzt werden.

Autostart-Funktion



Palette Autostart

Die Autostart-Funktion ermöglicht das automatische Laden und ggfls. auch Starten der zuletzt geöffneten Datei nach dem Programmstart. Unter *Startoptionen* kann gewählt werden, ob nur ein Laden der Datei oder auch ein Starten als Standard- bzw. Endlossimulation erfolgen soll. Wird BORIS mit Aufrufparametern gestartet, bleibt die Autostart-Funktion ohne Wirkung.

Betriebsartensteuerung

Die Steuerung der Simulation erfolgt zweckmäßigerweise über die entsprechenden Schaltflächen der System-Toolbar, kann bei Bedarf aber auch über die jeweiligen Optionen des Untermenüs SIMULATION vonstatten gehen. Nachfolgende Tabelle gibt zunächst eine Übersicht über die entsprechenden Optionen.

Symbol	Menüoption SIMULATION	Funktion
	START	Startet die Standard-Simulation

	STOP	Stoppt die Simulation
	PAUSE | EINZEL-SCHRITTMODUS	Aktiviert bzw. deaktiviert den Einzelschrittmodus
	EINZELSCHRITT AUSFÜHREN	Führt einen Einzelschritt aus
	START ENDLOSSIMULATION	Startet eine Endlossimulation
	BREAKPOINT SETZEN...	Setzt einen Breakpoint
	BREAKPOINT LÖSCHEN	Löscht einen Breakpoint

Starten der Standard-Simulation

In der Standard-Betriebsart wird die Simulation über die Menüfolge SIMULATION | START bzw. die System-Toolbar gestartet. BORIS überprüft nun zunächst, ob eine algebraische Schleife vorliegt oder einer der vorhandenen Systemblöcke unzulässig parametriert wurde. Gegebenenfalls wird eine entsprechende Warnung ausgegeben.

Eingänge von Systemblöcken können für die Simulation offen bleiben; sie werden in der Regel zu null bzw. in Sonderfällen auf andere Vorgabewerte gesetzt. Dadurch ist es beispielsweise bei speziellen Simulationsaufgaben (z. B. Simulation eines freien Systems mit vorgegebenen Anfangswerten) möglich, die Simulation zu starten, ohne zusätzlich einen Generator mit der Ausgangsgröße null einzusetzen.

Ist die Fehlerprüfung negativ verlaufen, so wird die Simulation gestartet. Während der Simulation wird die bereits verstrichene Simulationsdauer in der Statuszeile des Hauptfensters protokolliert, sofern diese Option aktiviert wurde.

In der Standard-Betriebsart wird die Simulation bis zu der über die Simulationsdauer vorgegebenen Zeit durchgeführt, sofern nicht zuvor ein Breakpoint erreicht wurde oder ein Benutzerabbruch erfolgte.

Endlossimulation

Im Gegensatz zur Standard-Simulation wird die vorgegebene Simulationsdauer bei einer Endlossimulation ignoriert. Die Simulation kann daher nur vom Anwender (bzw. ggf. auch über einen SIMCANCEL-Block) abgebrochen werden.

Einzelschrittsimulation

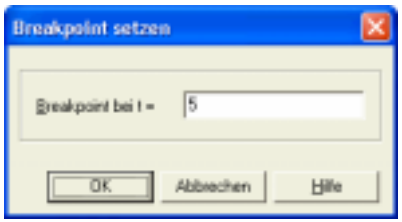
Der Anwender kann jederzeit während oder auch bereits vor der Simulation in den sogenannten *Einzelschrittmodus* wechseln. Wird der Einzelschrittmodus aktiviert, so erfolgt eine Unterbrechung der Simulation (Pause) und es kann im folgenden jeder Simulationsschritt einzeln freigegeben werden, bis der Einzelschrittmodus wieder verlassen wird. Diese Betriebsart eignet sich damit insbesondere für ein detailliertes Nachvollziehen bestimmter Simulationsbereiche. Der Einzelschrittmodus kann auch durch das Setzen von Breakpoints oder einen SIMCANCEL-Block automatisch aktiviert werden.

Abbrechen der Simulation

Die Simulation kann vom Anwender jederzeit vor ihrem regulären Ende abgebrochen werden. Dazu dient die Menüoption SIMULATION | STOP. Ist der Einzelschrittmodus aktiviert, so bleibt er dies auch nach Abbruch der Simulation!

Arbeiten mit Breakpoints

Durch das Setzen von sogenannten *Breakpoints* kann ein automatisches Umschalten in den Einzelschrittmodus zu einem festgelegten Zeitpunkt erfolgen.



Setzen eines Breakpoints (hier für t = 5)

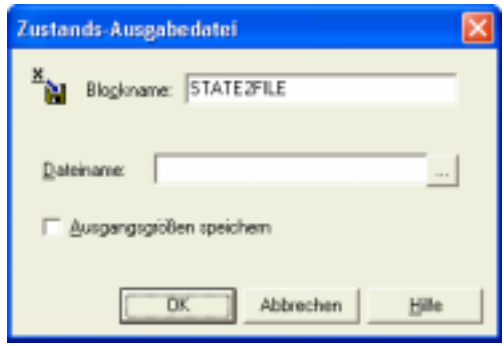
Erreicht die Simulation den vorgegebenen Zeitpunkt, erfolgt eine automatische Umschaltung in den Einzelschrittmodus. Mit Hilfe dieser Funktion lässt sich damit ein vorgegebener Zeitpunkt exakt anfahren, ohne dass die Simulation "per Hand" (und damit i. a. recht ungenau) an dieser Stelle unterbrochen wird.

Systemzustand in Anfangswerte übernehmen

Über die Option SIMULATION | SYSTEMZUSTAND IN ANFANGSWERTE ist es möglich, den aktuellen Zustand aller dynamischen Systemblöcke in deren Anfangswerte zu übernehmen, um z. B. zu einem späteren Zeitpunkt eine neue Simulation aus diesem Arbeitspunkt heraus fortzuführen. Dabei werden die aktuellen Zustandsgrößen der Blöcke in die Anfangswerte umkopiert. Diese Option ist in der aktuellen Version allerdings nur für Systemblöcke der obersten Hierarchieebene (d. h. der aktuell geladenen Systemdatei) verfügbar; die Anfangswerte von dynamischen Blöcken in Superblöcken können auf diese Weise nicht gesetzt werden! Die entsprechenden Superblöcke müssen daher ggf. vor der Simulation aufgesplittet werden.

Speichern und Laden von Systemzuständen

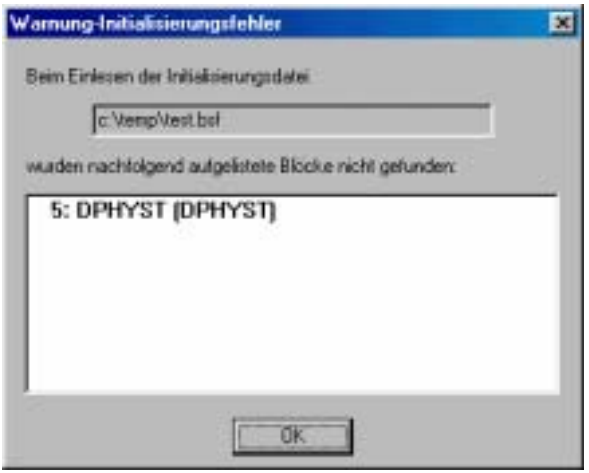
Soll der Zustand einer komplexeren Struktur (u. U. mit einer Vielzahl von Superblock-Ebenen) gespeichert werden, so kann dazu der Blocktyp *STATE2FILE* (Zustands-Ausgabedatei) benutzt werden. Dieser Block besitzt keine Ein- oder Ausgänge, sondern er sorgt dafür, dass die Zustandsgrößen nahezu aller in der Struktur vorhandenen Systemblöcke (auch derer in Superblöcken) bei Beendigung der Simulation in einer frei wählbaren BSF-Datei (*BORIS State File*) gespeichert werden. Zusätzlich dazu können auf Wunsch auch die Ausgangsgrößen des Blocks gespeichert werden.



Parameterdialog des STATE2FILE-*Blocks.*

Eine auf diese Weise erzeugte BSF-Datei kann dann später mit Hilfe des *FILE2STATE*-Blocktyps zur Initialisierung der Simulation benutzt werden. Dabei werden alle ggf. innerhalb des Parameterdialogs eingelesenen Anfangswerte von den in der BSF-Datei vorgefundenen Werten überschrieben.

Die Identifikation der einzelnen Systemblöcke bezüglich der BSF-Datei wird von BORIS über den Blockindex und den Blocktyp vorgenommen. Eine mit einem STATE2FILE-Block erzeugte BSF-Datei kann daher später über einen FILE2STATE-Block nur dann zur Initialisierung benutzt werden, wenn die Systemstrukturen beim Erzeugen und Wiedereinlesen der BSF-Datei identisch sind. Entdeckt BORIS beim Einlesen einer BSF-Datei Inkonsistenzen, so wird der Anwender über eine entsprechende Warnmeldung darauf aufmerksam gemacht (siehe nachfolgende Bildschirmgrafik).

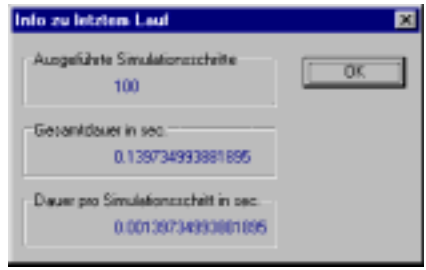


Warnmeldung beim Auftreten von Inkonsistenzen zwischen Systemstruktur und BSF-Datei. Hier wurde nach dem Erzeugen der BSF-Datei ein Block vom Typ DPHYST hinzugefügt.

Aus diesem Grund sollte der STATE2FILE-Block zum Erzeugen der BSF-Datei der Systemstruktur grundsätzlich als *letzter* Block hinzugefügt werden; nur so ist gewährleistet, dass bei einem späteren Löschen dieses Blocks die Reihenfolge der übrigen Blöcke (und damit deren Blockindizes) nicht geändert wird. Alternativ dazu kann der STATE2FILE-Block auch passiv gesetzt werden, statt ihn zu löschen.

Informationen zu letztem Lauf

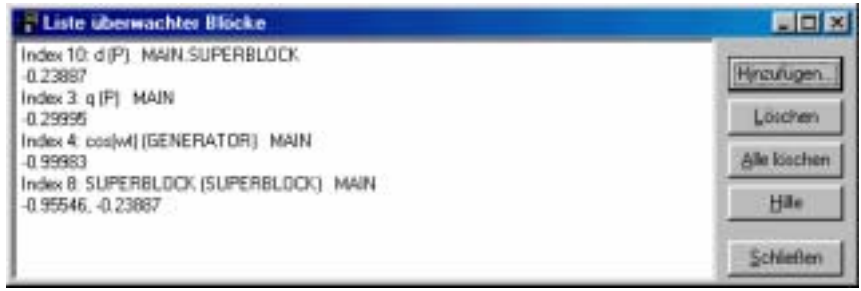
Über die Option SIMULATION | INFO ZU LETZTEM LAUF... können Informationen zum vorhergehenden Simulationslauf angefordert werden. Diese Informationen sind speziell im Zusammenhang mit Echtzeitsimulationen von Interesse, da sie Hinweise auf eine kleinstzulässige Simulationsschrittweite enthalten.



Info-Dialog zu vorangegangenem Simulationslauf

Überwachte Blöcke

Über die Option ANSICHT | ÜBERWACHTE BLÖCKE bietet BORIS ein Anzeigefenster, in dem während der Simulation alle Ausgangssignale beliebiger Blöcke angezeigt werden können. Der Vorteil dieses Fensters liegt insbesondere darin, dass mit seiner Hilfe auch Blöcke in Superblöcken auf einfache Weise beobachtet werden können.



Das BORIS-Fenster zur Überwachung von Blöcken

Jeder Block wird in zwei Zeilen dargestellt. Die jeweils obere Zeile enthält die Blockspezifikation, bestehend aus Blockindex, Blockname, Blocktitel und Blockebene. Die untere Zeile enthält jeweils alle aktuellen Ausgangssignale des Blocks, durch Kommata getrennt. Über die Schaltfläche *Hinzufügen* erscheint ein Dialog, über den der zu beobachtende Block ausgewählt werden kann. Über die Schaltfläche *Löschen* bzw. *Alle löschen* können einzelne oder alle Einträge gelöscht werden.

Hinweis: Die Identifikation der Blöcke innerhalb von BORIS wird über den Blockindex vorgenommen. Das hat zur Folge, dass nach dem Löschen von Blöcken aus der Systemstruktur einzelne Listeneinträge ungültig werden kön-

nen. Derartige Einträge werden beim Starten der nächsten Simulation automatisch von BORIS aus dem Anzeigefenster gelöscht.

Online-Parameteränderungen

Auch Parameteränderungen von Systemblöcken sind in der Regel während der Simulation möglich (Ausnahme: Systemblöcke, die über Dateien parametriert werden, wie z. B. *File-Input*). Dazu wird der gewünschte Systemblock wie gewohnt durch einen Doppelklick angewählt und die Änderungen vorgenommen. Nach Schließen des Parameterdialogs werden alle Änderungen sofort wirksam und die Simulation wird fortgeführt. Änderungen der Systemstruktur (z. B. Löschen von Blöcken oder Verbindungen) sind während der Simulation nicht möglich.

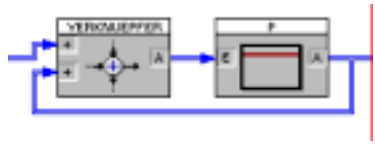
Was tun bei Algebraischen Schleifen?

Nachdem Sie einen Simulationslauf gestartet haben, sortiert BORIS alle Systemblöcke intern zunächst in eine für die Simulation geeignete Reihenfolge. Dies ist erforderlich, um unabhängig von der Reihenfolge, in der die Blöcke eingefügt wurden, immer reproduzierbare, korrekte Ergebnisse zu erhalten. Dieser Sortiervorgang schlägt jedoch fehl, wenn sich innerhalb Ihrer Simulationsstruktur eine sogenannte *Algebraische Schleife* befindet. Dies ist eine geschlossene Struktur, also eine Struktur mit Rückführung (Schleife), bei der sich innerhalb dieser Schleife nur Systemblöcke befinden, die nicht verzögernd wirken, d. h. bei denen sich eine Änderung am Eingang noch im selben Simulationsschritt am Ausgang bemerkbar macht. Damit BORIS die Blöcke sortieren kann, muss sich innerhalb einer Schleife also immer mindestens ein Block mit Verzögerung befinden. Dazu zählen folgende Systemblocktypen:

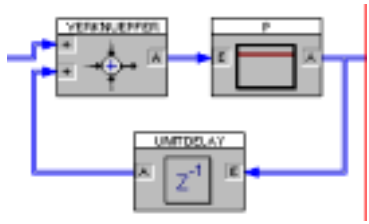
- PT_1, PT_2, PT_1T_2, PT_n - Glieder
- Integrierer und Totzeitglieder
- Allpassglieder
- Übertragungsfunktionen mit Zählergrad < Nennergrad
- Einheitsverzögerungen

Tritt innerhalb Ihrer Struktur eine algebraische Schleife auf, macht BORIS Sie durch eine entsprechende Meldung darauf aufmerksam und zeigt die für die Schleife "verantwortlichen" Blöcke an. Sie sollten in diesem Fall Ihr Modell zunächst überprüfen, da algebraische Schleifen in der Regel auf Modellie-

rungsfehler hinweisen (in der Realität gibt es keine algebraischen Schleifen!). Wollen Sie die Struktur dennoch simulieren, können Sie zur Abhilfe einen der oben aufgeführten Verzögerungsblöcke in die Schleife einfügen; am besten eignet sich dazu eine Einheitsverzögerung, bei der das Eingangssignal genau um einen Takt verzögert wird oder ein PT_1-Glied mit einer kleinen Zeitkonstanten. Nachfolgende Grafiken verdeutlichen das Prinzip.



Beispiel für eine Algebraische Schleife...



...und ihre Aufhebung durch Einfügen einer Einheitsverzögerung

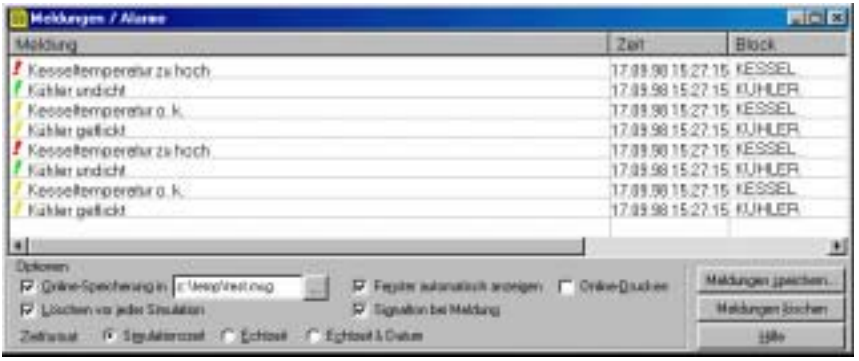
Was Sie sonst noch wissen sollten

Verwaltung von Alarmen/Meldungen

BORIS besitzt eine leistungsfähige Meldungs- bzw. Alarmverwaltung, mit der Sie während der Simulation anfallende Meldungen/Alarme zentral erfassen und anzeigen können. Dazu dient ein Meldungsfenster, das über die Option ANSICHT | MELDUNGEN/ALARME jederzeit angezeigt werden kann. Zur Generierung von Meldungen fügen Sie einfach an entsprechenden Stellen einen Meldungs-Block ein. Jede Meldung wird im Klartext mit dem Namen des

entsprechenden Meldungsblocks sowie Simulationszeit oder Echtzeit/Datum beim Auftreten der Meldung angezeigt und kann darüber hinaus auch akustisch in Form einer WAV-Datei unterlegt werden. Das Meldungsfenster bietet neben der reinen Anzeigefunktion folgende Optionen:

- Durch Aktivierung von *Online-Speicherung* werden Meldungen automatisch in der im nachfolgenden Eingabefeld spezifizierten Datei gespeichert.
- Vor dem Beginn einer neuen Simulation können die angezeigten Meldungen automatisch gelöscht werden (Option *Löschen vor jeder Simulation*).
- Das Meldungsfenster kann zu Beginn der Simulation automatisch angezeigt werden (Option *Fenster autom. anzeigen*; Anzeige erfolgt nur, wenn System mindestens einen Meldungs-Block enthält!).
- Beim Eintreffen einer Meldung kann ein Signalton erzeugt werden (Option *Signalton bei Meldung*).
- Alle angezeigten Meldungen können manuell gelöscht und in einer beliebigen ASCII-Datei gespeichert werden (Schalter *Meldungen löschen* bzw. *Meldungen speichern.....*).



Verwaltungsfenster für Meldungen/Alarme

Die Priorität der Meldung wird durch die Farbe des Ausrufezeichens vor dem Meldungstext gekennzeichnet. Dabei weist gelb Meldungen mit niedriger Priorität, grün Meldungen mit mittlerer Priorität und rot Meldungen mit hoher Priorität aus. Weitere Einzelheiten entnehmen Sie bitte der Beschreibung des Meldungsblocks im Abschnitt *Die BORIS-Systemblock-Bibliothek*.

Verriegeln des Hauptfensters

Um z. B. während länger andauernder Simulationen oder Messwerterfassungen unbefugte Zugriffe zu verhindern, kann das BORIS-Hauptfenster verriegelt werden. Dies wird über die Menüfolge ANSICHT | VERRIEGELN bzw. die Schaltfläche bewerkstelligt. BORIS fragt anschließend nach einem Passwort, welches später zur Wiederherstellung des Hauptfensters dient.

Eingabe des Passworts zum Verriegeln des Hauptfensters

Das Passwort muss zur Sicherung vor Fehleingaben in einem zweiten Eingabefeld bestätigt werden. Anschließend wird das Hauptfenster zum Symbol verkleinert und kann danach nur durch erneute Eingabe des Passworts wieder regeneriert werden.

Eingabe des Passworts zum Entriegeln des Hauptfensters

Zugriffsschutz für Systemdateien

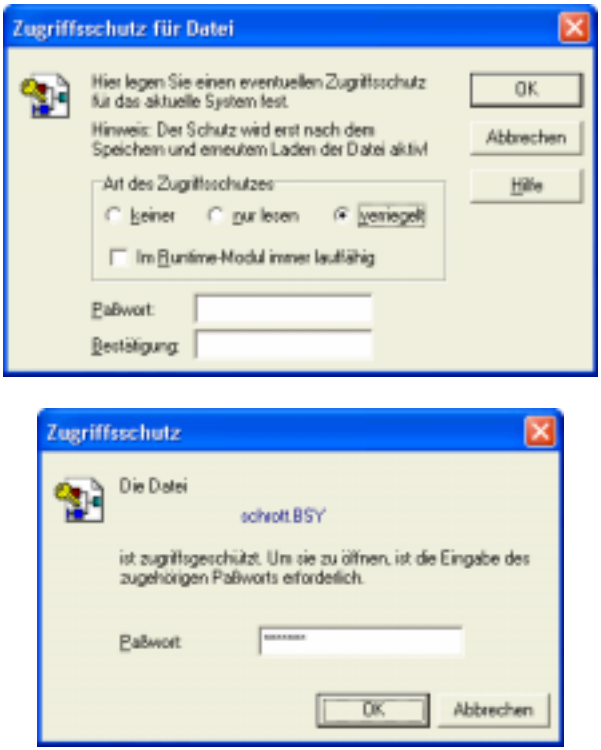
BORIS-System- und Superblockdateien können mit einem Zugriffsschutz versehen werden, der ein unbefugtes Modifizieren oder Öffnen der Datei verhindert. Dazu dient die Menüoption DATEI | ZUGRIFFSSCHUTZ... Es sind drei Stufen des Zugriffsschutzes verfügbar:

keiner Keine Einschränkungen für Dateizugriff

nur lesen Eine Modifikation des Systems ist nur nach Eingabe des korrekten Passworts möglich. Wird beim Öffnen der Datei kein Passwort bzw. ein falsches Passwort angegeben, kann das System nur gelesen und simuliert werden.

verriegelt Ein Öffnen des Systems ist nur nach Eingabe des korrekten Passworts möglich.

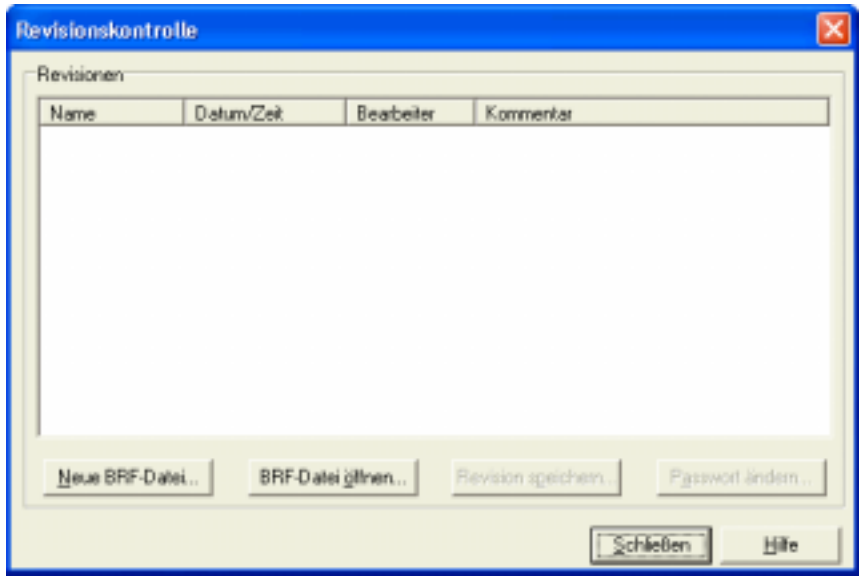
Das eingegebene Passwort muss zur Sicherung vor Fehleingaben im unteren Eingabefeld noch einmal bestätigt werden. Beachten Sie bitte, dass ein eventueller Zugriffsschutz erst nach dem Speichern beim folgenden Ladeversuch aktiviert wird.



Festlegen des Zugriffsschutzes für System- bzw. Superblockdateien (oben) und Passwort-Abfrage beim Öffnen einer Datei mit Zugriffsschutz (unten)

Das Revisions-Kontrollsystem von BORIS

Für komplexere Projekte besitzt BORIS ein leistungsfähiges und komfortables Revisions-Kontrollsystem, mit dem sich die unterschiedlichen Entwicklungsstadien eines Projektes übersichtlich zusammenfassen und bei Bedarf wiederherstellen lassen. Das Revisions-Kontrollsystem wird über die Menüfolge DATEI | REVISIONSKONTROLLE... aufgerufen.



Dialog zur Revisionskontrolle

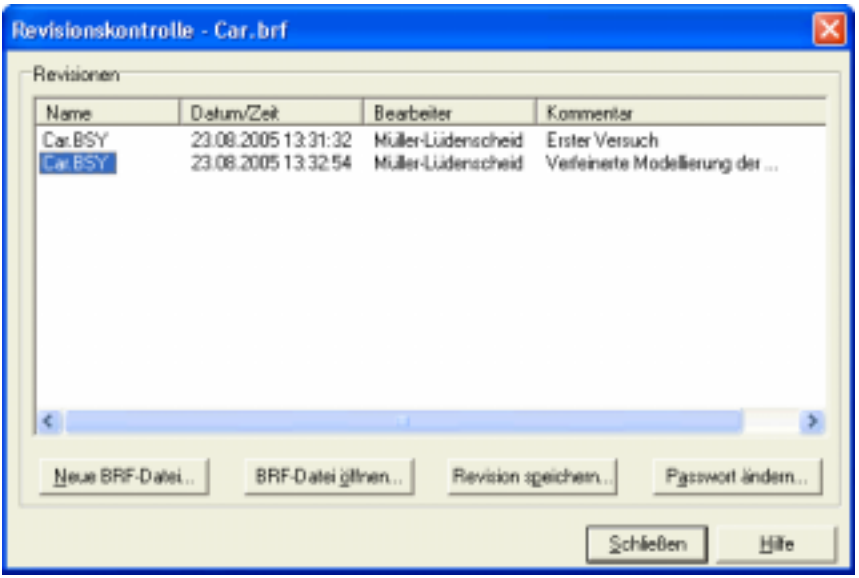
Basis des Revisions-Kontrollsystems bilden BORIS-Revisionsdateien (Dateiendung BRF für *BORIS Revision Files*). Eine solche Revisionsdatei enthält dabei beliebig viele System- oder Superblockdateien in verschlüsselter Form und auf Wunsch durch ein Kennwort geschützt. Die enthaltenen Dateien spiegeln dabei in der Regel unterschiedliche Entwicklungsstadien eines Projektes wieder und können bei Bedarf aus der BRF-Datei wieder extrahiert werden. Jede abgelegte Datei enthält Informationen über das Dateidatum, den Bearbeiter sowie einen Kommentar zur besseren Kenntlichmachung.

Soll ein Projekt über das Revisions-Kontrollsystem verwaltet werden, so muss dazu zunächst über die Schaltfläche *Neue BRF-Datei...* eine neue BRF-Datei angelegt werden. Über die Schaltfläche *BRF-Datei öffnen...* kann eine bereits vorhandene Revisionsdatei geöffnet werden. Eine einmal geöffnete Revisions-

datei bleibt auch nach Verlassen des Dialogs geöffnet und steht beim erneuten Aufruf daher unmittelbar wieder zur Verfügung.

Beim Anlegen einer neuen Revisionsdatei erfolgt automatisch die Abfrage eines Kennworts, mit dem die Datei gegen unbefugten Zugriff geschützt werden soll. Bei Bedarf kann dieser Kennwortschutz durch Eingabe eines Leerstrings deaktiviert werden. Das Kennwort kann später jederzeit über die *Passwort ändern*...-Schaltfläche geändert werden.

Nach dem Anlegen einer neuen oder dem Öffnen einer bereits existierenden Revisionsdatei kann nun die jeweils aktuell in BORIS bearbeitete Datei in die Revisionsdatei übernommen werden. Dazu dient die Schaltfläche *Revision speichern*. Nachfolgende Grafik zeigt den Dialog nach Einfügen von zwei Versionen der Datei *car.bsy*.

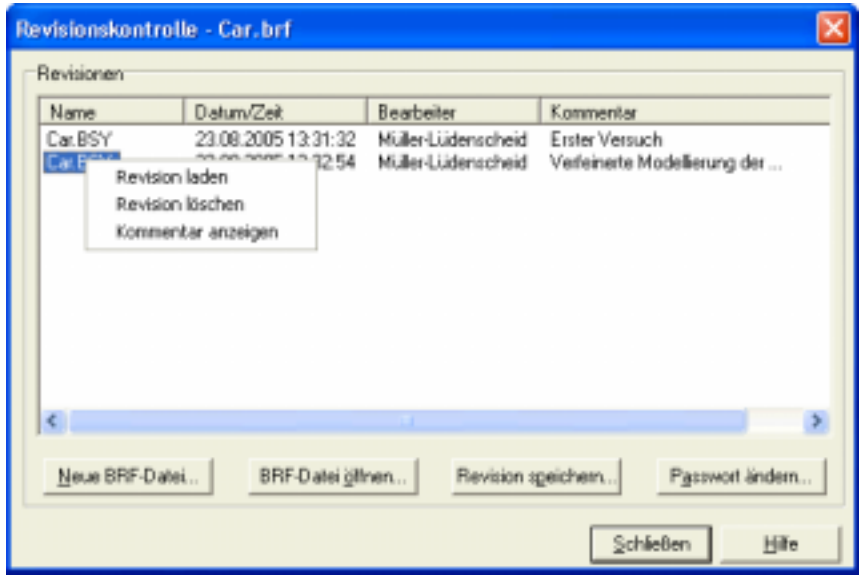


Dialog mit zwei Versionen der Datei car.bsy

Durch Anklicken eines Dateinamens (linke Spalte der Liste) mit der rechten Maustaste erscheint ein Kontextmenü, über das weitere Optionen angeboten werden:

- Laden einer Revision aus der Revisionsdatei in BORIS
- Löschen einer Revision aus der Revisionsdatei
- Anzeige des kompletten Kommentars zu einer Revision

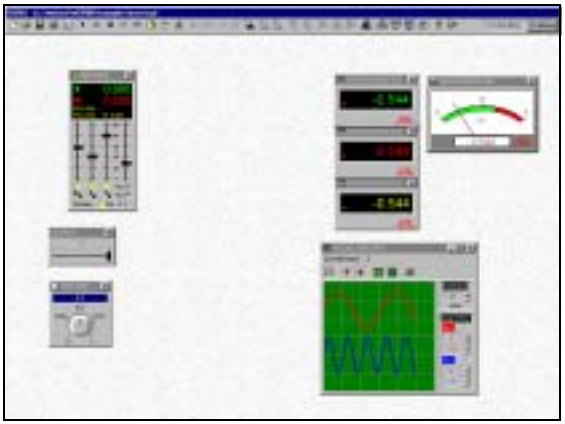
Das Laden einer Revision kann alternativ auch durch einen Doppelklick auf den Dateinamen erfolgen, das Löschen durch Betätigung der <Del>-Taste.



Kontextmenü der Revisionskontrolle

Wechsel des Anzeigemodus

Für bestimmte Anwendungsfälle von BORIS (z. B. Messdatenerfassungen) kann es wünschenswert sein, das eigentliche Hauptfenster mit der Systemstruktur ausblenden zu können und nur die Aktions- bzw. Anzeigeblöcke auf dem Bildschirm erscheinen zu lassen. BORIS bietet zu diesem Zweck einen alternativen Anzeigemodus an, der über ANSICHT | ANZEIGEMODUS WECHSELN aktiviert werden kann. Das BORIS-Hauptfenster wird dabei bis zu einer modifizierten System-Toolbar verkleinert und mit einem neutralen Hintergrund versehen. Über die *Schließen*-Schaltfläche kann zurück in den Standardmodus gewechselt werden.



Alternativer Anzeigemodus

Benutzerdefinierte Einstellungen

BORIS erlaubt Ihnen über die Menüoption OPTIONEN | ANPASSEN... die Vorgabe einer Reihe von benutzerdefinierten Einstellungen.

Anzeige-Optionen

Über die Palette *Anzeige* sind einige Optionen erreichbar, die insbesondere die Bildschirmanzeige betreffen:

Option	**Bedeutung**
Darstellung	Darstellungsart von Verbindungen
Verbindungen von/zu passiven Blöcken grau	Darstellungsart von Verbindungen von/zu passiv gesetzten Blöcken
Knotenpunkte auf Bildschirm anzeigen	Legt fest, ob die Knotenpunkte an Abzweigungen von Verbindungen auf dem Bildschirm dargestellt werden.
Farbe für neue Verbindungen ändern	Ermöglicht die Wahl einer neuen voreingestellten Farbe für zukünftig eingefügte Blöcke. Diese Option beeinflusst nicht bereits vorhandene Verbindungen!
Standard-Blockgröße	Ermöglicht die Wahl einer neuen voreingestellten Größe für zukünftig eingefügte Blöcke. Diese Option beeinflusst nicht bereits vorhandene Böcke!
Blockindex anzeigen	Ist diese Option aktiviert, wird im Blocktitel eines Blocks neben dem Blocknamen auch der Blockindex angezeigt.
Standard-Fontgröße für Textblöcke	Ermöglicht die Wahl einer neuen voreingestellten Schriftgröße für zukünftig eingefügte Textblöcke. Diese Option beeinflusst nicht bereits vorhandene Textblöcke!
Bildschirmhinter-grund-Farbe ändern	Über diese Schaltfläche kann die Farbe für den Hintergrund der BORIS-Zeichenfläche modifiziert werden.

Raster

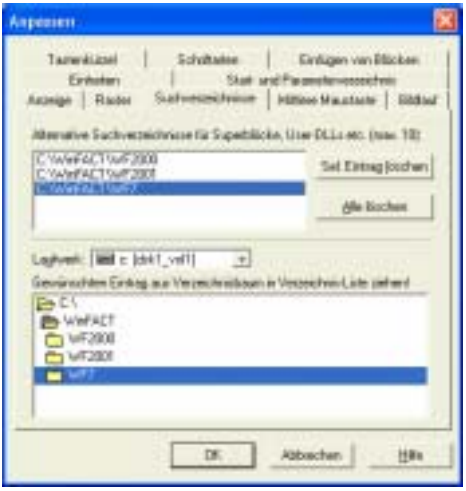
Die Optionen dieser Palette haben nachfolgende Bedeutung.

Option	Bedeutung
Raster sichtbar	Schaltet die Anzeige des Bildschirmrasters zu bzw. ab
An Raster ausrichten	Aktiviert bzw. deaktiviert die Ausrichtung der Systemblöcke am Raster. Die Ausrichtung ist unabhängig davon, ob das Raster sichtbar ist.
Rasterweite in x-Richtung	Legt den horizontalen Abstand zwischen zwei Rasterpunkten in Pixeln fest
Rasterweite in y-Richtung	Legt den vertikalen Abstand zwischen zwei Rasterpunkten in Pixeln fest

Startverzeichnis

Auf dieser Palette legen Sie fest, in welches Verzeichnis BORIS nach dem Starten zunächst wechseln soll (welches dann also z. B. beim Öffnen von Dateien im Dateidialog als voreingestellt erscheint). Dies kann einerseits das Verzeichnis sein, welches beim vorangegangenen Verlassen von BORIS zuletzt in Benutzung war, andererseits kann auch ein beliebiges, fest vorgegebenes Verzeichnis gewählt werden.

Suchverzeichnisse



BORIS erlaubt Ihnen über die Menüoption OPTIONEN | ANPASSEN..., Palette *Suchverzeichnisse*, die Vorgabe von bis zu zehn Suchpfaden. In diesen Pfaden werden (in der Reihenfolge des Eintrags) alle dateibasierten Blöcke (also z. B. Superblöcke oder User-Blöcke) gesucht, sofern die vom Anwender vorgegebenen Namen keinen oder aber einen nicht vorhandenen Pfad enthalten. Einen neuen Eintrag fügen Sie einfach ein, indem Sie das gewünschte Verzeichnis aus dem Verzeichnisbaum im unteren Bereich des Dialogs per Drag & Drop in die obere Liste ziehen und dort fallenlassen. Über die Schaltfläche *Sel. Eintrag löschen* kann der gerade selektierte Eintrag gelöscht werden, über *Alle löschen* werden alle Einträge gelöscht.

Die Definition von Suchverzeichnissen ist insbesondere dann angebracht, wenn komplexe Systemstrukturen mit vielen Superblöcken, User-DLLs usw. auf einen anderen Rechner mit anderer Verzeichnisstruktur portiert werden sollen. Es ist dann nicht erforderlich, alle Pfadangaben in den entsprechenden Systemblöcken mühsam per Hand zu ändern, sondern die Definition eines oder einiger weniger Suchverzeichnisse reicht aus.

Beispiel: Es sei auf Rechner A eine BSY-Datei im Verzeichnis C:\WINFACT erstellt worden, die diverse Superblöcke im Unterverzeichnis C:\WINFACT\SB enthalte. Diese Simulationsstruktur soll nun auf Rechner B portiert werden, auf dem BSY-Dateien gewöhnlich im Verzeichnis C:\WF und Superblockdateien im Unterverzeichnis C:\WF\SUPER liegen. Dann reicht es aus, diese beiden Pfadnamen auf Rechner B als Suchverzeichnisse anzugeben und die Simulation kann unmittelbar gestartet werden. Beim nächsten Abspei-

chern des Systems auf Rechner B werden alle Dateinamen dann außerdem automatisch angepasst.

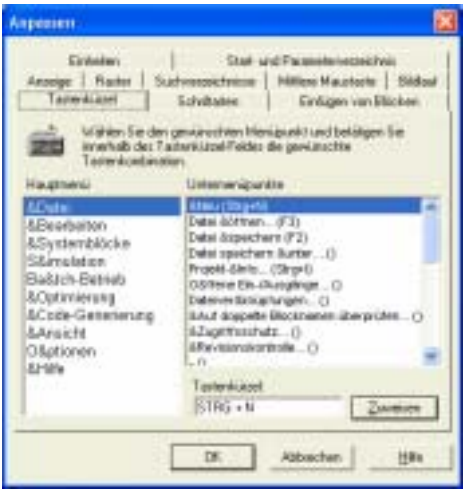
Konfigurierung der mittleren Maustaste

Diese Palette erlaubt Anwendern mit einer Drei-Tasten-Maus die Konfigurierung der mittleren Maustaste auf eine bestimmte Funktion.

Bildlauf

Diese Palette ermöglicht die Anpassung der Bildlaufgeschwindigkeit (Scrollgeschwindigkeit) beim Verschieben von Blöcken etc. an die benutzte Rechnerkonfiguration, insbesondere an die verwendete Grafikkarte. Wird beim Verschiebevorgang zusätzlich die <Shift>-Taste gedrückt, erfolgt der Bildlauf außerdem mit doppelter Geschwindigkeit.

Tastenkürzel



Über diese Palette lassen sich die Tastenkürzel (Shortcuts) für sämtliche Befehle des BORIS-Hauptmenüs an die eigenen Gewohnheiten anpassen. Dazu wird zunächst der gewünschte Menüpunkt über die beiden Listen *Hauptmenü* und *Untermenüpunkte* ausgewählt. Im Eingabefeld *Tastenkürzel* kann dann durch die direkte Einagbe der gewünschten Taste oder Tastenkombination (z. B. <Strg> <A>) und anschließende Betätigung der *Zuweisen*-Schaltfläche die Zuordnung getroffen werden.

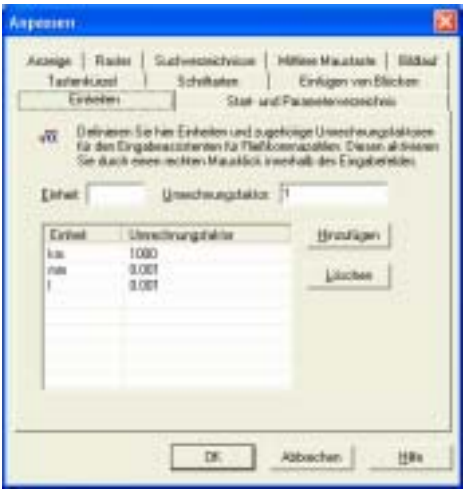
Schriftarten

Je nach Rechner-Betriebssystem und installierten Schriftarten können die Blocktitel, die in der Schriftgröße 6 pt angezeigt werden, gelegentlich schlecht lesbar erscheinen. Daher kann über diese Palette die am besten geeignete Schriftart für die Blocktitel ausgewählt werden.

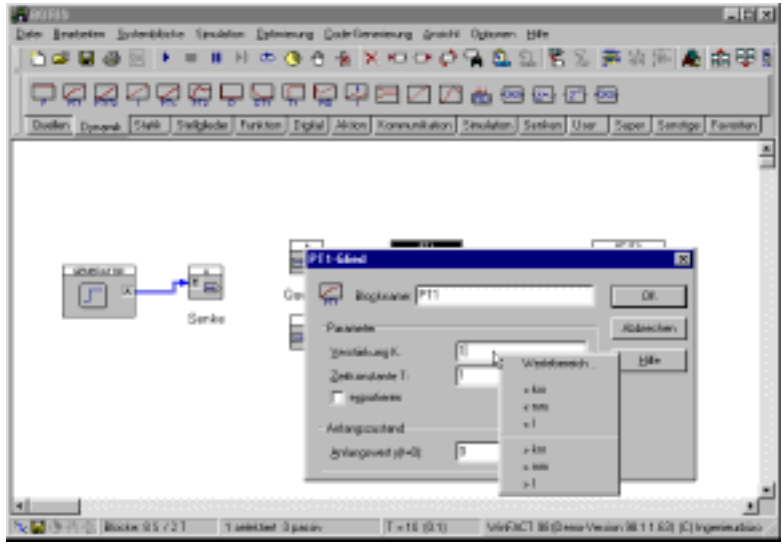
Einfügen von Blöcken

BORIS erlaubt das Einfügen neuer Blöcke sowohl über einen Einfachklick auf die entsprechende Schaltfläche der Systemblock-Toolbar (wobei der eingefügte Block dann automatisch plaziert wird), als auch per Drag & Drop. Beide Einfügeoptionen können über diese Palette getrennt voneinander aktiviert bzw. deaktiviert werden.

Einheiten



Um die Umrechnung von Größen, die in unterschiedlichen Einheiten vorliegen (z. B. m^3/s und l/h), besitzt BORIS einen integrierten Eingabeassistenten, der automatisch aktiviert wird, wenn innerhalb des Eingabefeldes für eine Fließkommazahl die rechte Maustaste gedrückt wird. In dem daraufhin erscheinenden Popup-Menü kann auf die eingegebene Zahl direkt ein einheitenspezifischer Umrechnungsfaktor oder sein Kehrwert angewendet werden. Über die Palette *Einheiten* können diese Einheiten zusammen mit ihren Umrechnungsfaktoren konfiguriert werden. Dazu wird im Eingabefeld *Einheit* die Einheit und im Feld *Umrechnungsfaktor* der zugehörige Umrechnungsfaktor angegeben. Über die Schaltfläche *Hinzufügen* wird die Einheit in die Liste aufgenommen. Der zugehörige Kehrwert wird von BORIS automatisch erzeugt.

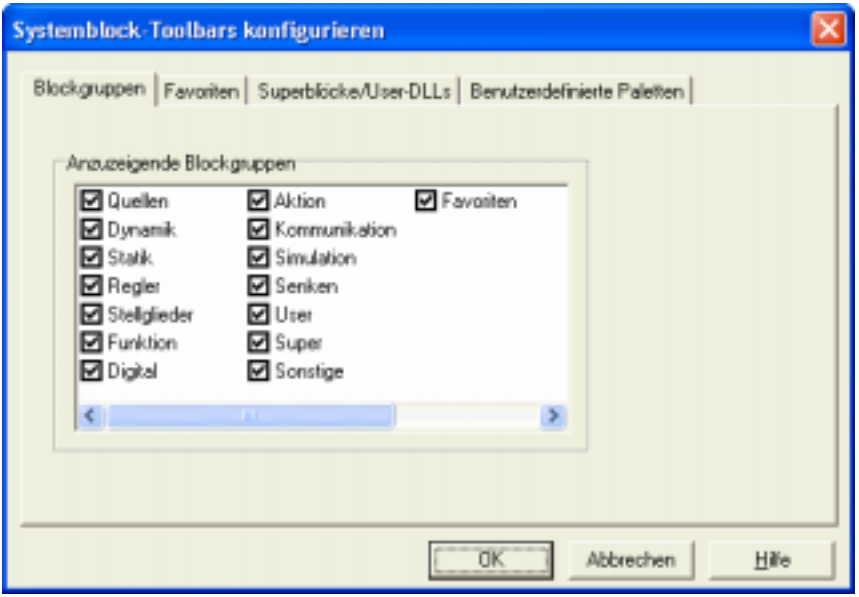


Aufruf des Eingabeassistenten für Fließkommazahlen

Konfigurierung der Systemblock-Toolbar

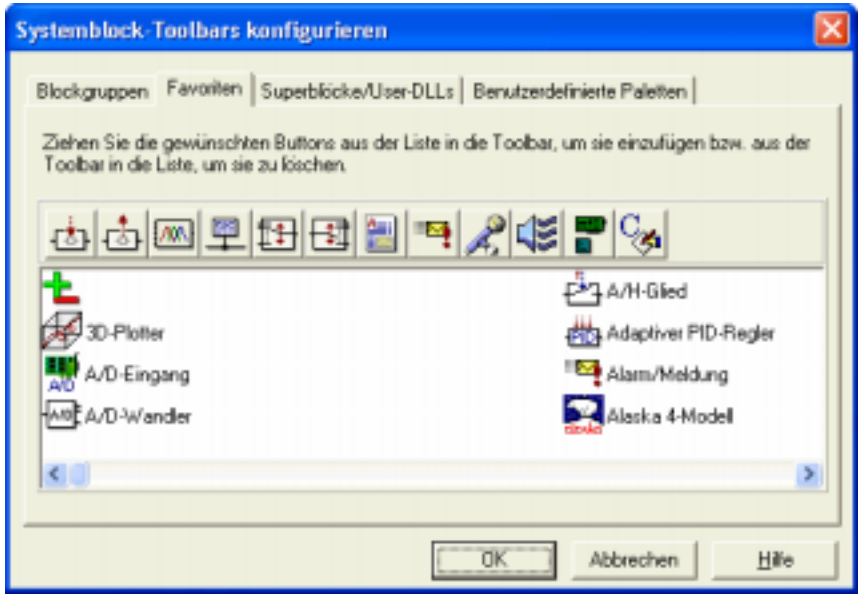
Die Systemblock-Toolbar kann vom Anwender an seine Bedürfnisse angepasst und um eigene Paletten ergänzt werden. Der zugehörige Dialog, der in vier Paletten aufgeteilt ist, wird über die Menüoption OPTIONEN | TOOLBARS | SYSTEMBLOCKTOOLBAR KONFIGURIEREN... verfügbar.

Anzuzeigende Blockgruppen



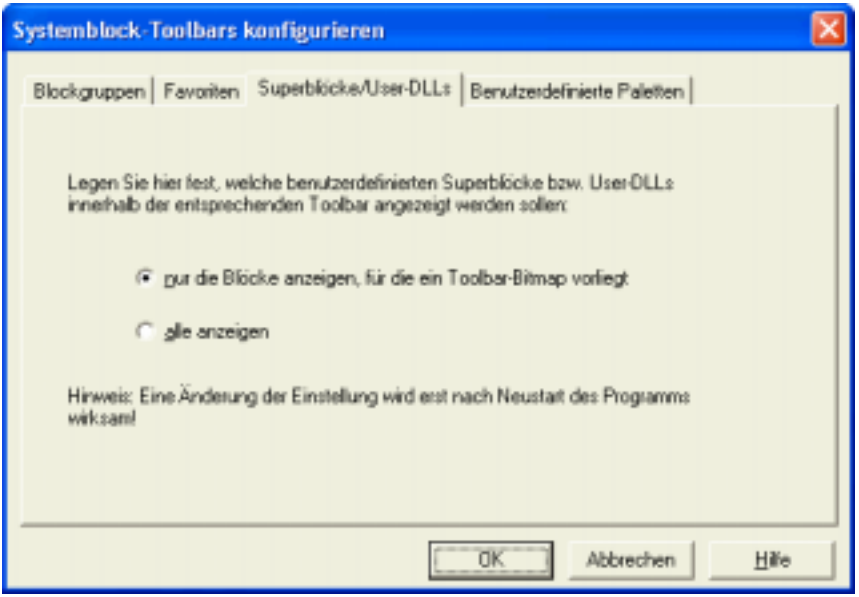
Die Systemblock-Toolbar ist aufgeteilt in mehrere Paletten, die jeweils zusammengehörige Blocktypen umfassen. Über die Palette *Blockgruppen* lassen sich einzelne Paletten der Systemblock-Toolbar bei Bedarf ausschalten. Standardmäßig sind sämtliche Paletten sichtbar.

Palette Favoriten



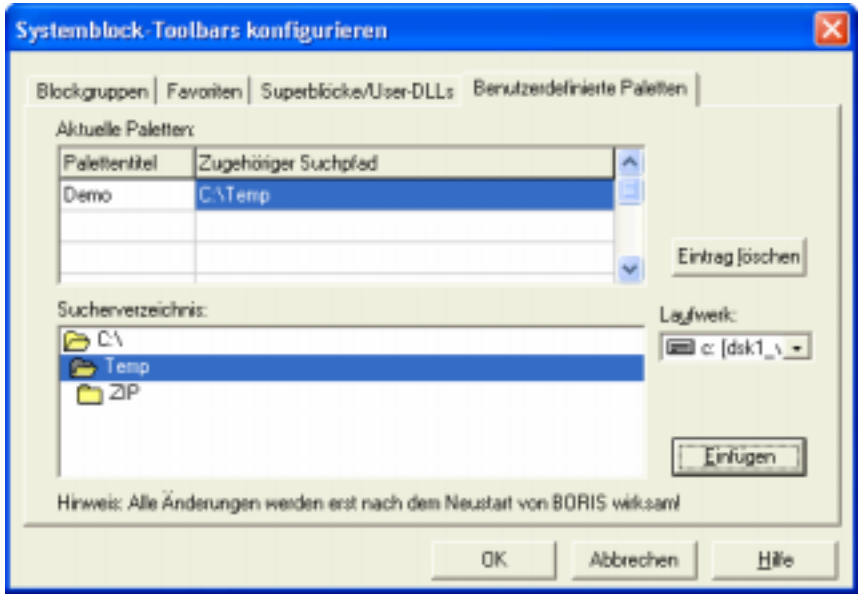
Die *Favoriten*-Palette der Systemblock-Toolbar kann vom Anwender über diese Dialogpalette frei konfiguriert werden. Dazu werden die gewünschten Blocktypen aus dem unteren Listenfenster einfach per Drag & Drop auf die darüberliegende Toolbar gezogen. Standardmäßig ist die *Favoriten*-Palette leer.

Superblöcke und User-DLLs



Die Paletten *User* bzw. *Super* der Systemblock-Toolbar enthalten neben jeweils „leeren“ Blöcken des entsprechenden Typs auch alle benutzerdefinierten Superblöcke bzw. User-DLLs, die BORIS im WinFACT 98-Programmverzeichnis oder in einem der Suchverzeichnisse findet. Über diese Dialogpalette wird festgelegt, ob dabei sämtliche benutzerdefinierten Blöcke aufgenommen werden, oder nur diejenigen, für die ein benutzerdefiniertes Toolbar-Bitmap gefunden wurde. Beachten Sie dabei, dass eine Änderung dieser Einstellung erst beim nächsten Aufruf von BORIS aktiv wird.

Benutzerdefinierte Paletten



Neben den Standard-Blocktypen-Paletten kann die Systemblock-Toolbar vom Anwender um eigene Paletten erweitert werden. Die auf diesen Paletten erscheinenden Blöcke sind dann jeweils alle User-DLLs bzw. Superblöcke, die BORIS innerhalb eines bestimmten Verzeichnisses findet. Um eine neue Palette einzufügen, gehen Sie wie folgt vor:

- Geben Sie in der Liste im oberen Dialogteil in der Spalte *Palettentitel* den Namen ein, den Ihre Palette tragen soll (oben *Demo*).
- Wählen Sie im darunter liegenden Verzeichnisbaum das Verzeichnis, auf dem Ihre Palette basieren soll und fügen Sie den Verzeichnisnamen (oben *c:\temp*) in die Liste ein.

Beim nächsten Start von BORIS finden Sie dann die neue Palette in der Systemblock-Toolbar vor.

Starten von BORIS mit Aufrufparametern

BORIS kann optional mit Aufrufparametern gestartet werden, die ein automatisches Einlesen einer Datei und bei Bedarf auch einen automatischen Simulationsstart bewirken:

BORIS *Dateiname*	startet BORIS mit der Datei *Dateiname*.
BORIS *Dateiname* /S	startet BORIS, lädt automatisch die Datei *Dateiname* und startet die Simulation. Der Schalter */S* kann wahlweise auch vor dem Dateinamen stehen.
BORIS *Dateiname* /E	startet BORIS, lädt automatisch die Datei *Dateiname* und startet eine Endlossimulation. Der Schalter */E* kann wahlweise auch vor dem Dateinamen stehen.
BORIS *Dateiname* /O	startet BORIS, lädt automatisch die Datei *Dateiname* und startet eine Parameteroptimierung. Der Schalter */O* kann wahlweise auch vor dem Dateinamen stehen.
BORIS *Dateiname ...* /C	beendet BORIS automatisch nach Durchführung der Simulation bzw. Optimierung. Dieser Schalter ist nur in Verbindung mit den Schaltern */S* bzw. */O* sinnvoll.

Betrieb von BORIS als COM-Automatisierungsserver

BORIS stellt für eine Vielzahl von Funktionen und Parametern eine *IDispatch*-Schnittstelle zur Verfügung, die von anderen Windows-Programmen zur "Fernsteuerung" von BORIS angesprochen werden kann; BORIS arbeitet in dieser Betriebsart dann als so genannter *COM-Automatisierungsserver*. Die Schnittstelle besteht einerseits aus *Methoden*, über die bestimmte Funktionen von BORIS ausgelöst werden können, als auch aus *Properties* (Eigenschaften), über die BORIS-Einstellungen (Parameter) ausgelesen werden können. Der Name des Automatisierungsservers lautet *BORIS.BoAutoObject*. Nachfolgende Tabellen geben einen Überblick über die zur Verfügung stehenden Methoden und Properties der Schnittstelle.

Methode	Parameter	Rückgabe-wert	Funktion
LoadFile	*Filename* (BSTR)	-	Lädt die Datei mit dem Namen *Filename*
AddFile	*Filename* (BSTR)	-	Fügt die Datei mit dem Namen *Filename* zur bestehenden Struktur hinzu
Clear	-	-	Löscht das aktuelle Arbeitsblatt
StartSim	-	-	Startet eine Standard-Simulation
StartEndlessSim	-	-	Startet eine Endlossimulation
StopSim	-	-	Stoppt die Simulation
Break	-	-	Setzt die Simulation in den Pause- bzw. Einzelschrittmodus
SingleStep	-	-	Führt einen Einzelschritt aus
ShowWindows	-	-	Zeigt alle Anzeigefenster an
HideWindows	-	-	Minimiert alle Anzeigefenster
MinimizeRestore	-	-	Verkleinert das Hauptfenster auf Toolbarhöhe bzw. versetzt es wieder in den ursprünglichen Zustand
GetBlockOutput	*BlockIndex* (LONG), *OutIndex* (LONG)	VARIANT	Liefert für den Block mit dem Blockindex *Blockindex* den Wert des Ausgangs *OutIndex*
SetConstValue	*BlockIndex* (LONG), *Value* (DOUBLE)	-	Setzt für einen Block vom Typ *Konstante* mit dem BlockIndex *BlockIndex* den Parameterwert auf *Value*

Methoden der IDispatch-*Schnittstelle von BORIS*

Property	Typ	Bedeutung
Filename	BSTR	Name der aktuell geladenen Datei
Time	DOUBLE	Aktuelle Simulationszeit
DeltaT	DOUBLE	Simulationsschrittweite
TSimu	DOUBLE	Gesamt-Simulationsdauer
IsRealTimeSim	BOOL	Echtzeitsimulation ja/nein
IsSimulating	BOOL	Standard-Simulation läuft ja/nein
IsEndlessSimulating	BOOL	Endlossimulation läuft ja/nein
IsBreak	BOOL	Simulation befindet sich im Einzelschrittmodus ja/nein

Properties der IDispatch-*Schnittstelle von BORIS*

Nachfolgendes Quelltextfragment zeigt die Nutzung der Schnittstelle aus DELPHI; in diesem kleinen Beispiel wird BORIS auf Knopfdruck gestartet, die Datei *c:\temp\test.bsy* geladen und die Endlossimulation gestartet.

```
procedure TForm1.Button1Click(Sender: TObject);
var V: Variant;
begin
  // Objekt anlegen, d. h. BORIS öffnen
  V := CreateOLEObject('BORIS.BoAutoObject');
  // Datei laden
  V.LoadFile('c:\temp\test.bsy');
  // Endlossimulation starten
  V.StartEndlessSim;
end;
```

Nutzung der Schnittstelle aus DELPHI

Nachfolgendes Listing zeigt den äquivalenten Quelltext für Visual Basic:

```
Private Sub Command1_Click()
Dim V As Object
Set V = CreateObject("BORIS.BoAutoObject")
V.LoadFile ("c:\temp\test.bsy")
```



```
V.StartEndlessSim
End Sub
```

Nutzung der Schnittstelle aus Visual Basic

Im Unterverzeichnis *\VBAutoDemo* Ihrer WinFACT-Installation befindet sich ein komplettes Visual Basic-Projekt (incl. Quelltexten), das die Nutzung sämtlicher Funktionen des Automatisierungsservers demonstriert; nachfolgende Bildschirmgrafik zeigt das Demoprogramm nach dem Aufruf.

Hinweis: Damit BORIS als Automatisierungsserver angesprochen werden kann, muss der Server in der Windows-Registry einmalig registriert werden. Dies geschieht *automatisch* beim erstmaligen Aufruf von BORIS "per Hand"!

Visual Basic-Testprogramm VBAutoDemo

Die BORIS-Systemblock-Bibliothek

Arten von Systemblöcken

In den folgenden Abschnitten werden alle Module der BORIS-Systemblockbibliothek beschrieben. Die Systemblocktypen sind aufgeteilt in folgende Typklassen, die jeweils auf einer eigenen Palette der Systemblock-Toolbar zusammengefasst sind:

Quellen

In diese Gruppe gehören diejenigen Systemblocktypen, die Eingangssignale für das System liefern (z. B. Generator, Datei-Eingabe).

Dynamik

Hierunter fallen alle linearen und nichtlinearen dynamischen Systeme, beginnend beim einfachen P-Glied bis hin zu frei parametrierbaren Übertragungsfunktionen und Dgl.-Systemen.

Statik

Dies sind im wesentlichen nichtlineare Kennlinien- bzw. Kennfeldglieder (Beispiele: Begrenzer, Tote Zone).

Regler

Diese Blockgruppe enthält eine Vielzahl unterschiedlicher linearer und nichtlinearer sowie stetiger und schaltender Reglertypen.

Stellglieder

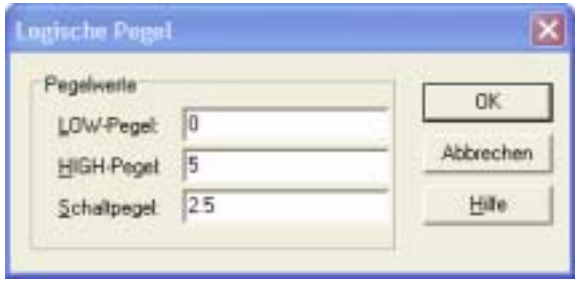
Diese Klasse enthält Blöcke, die das Verhalten realer industrieller Stellglieder nachbilden.

Funktion

Als Funktionsblöcke werden diejenigen Blocktypen bezeichnet, die nicht als Übertragungssysteme im herkömmlichen Sinne anzusehen sind. Hierzu gehören insbesondere Summierer und andere Verknüpfer mehrerer Eingangsgrößen.

Digital

Diese Klasse enthält solche Blöcke, deren Ausgangsgröße nur die binären Zustände high (logisch 1) bzw. low (logisch 0) annehmen kann. Die entsprechenden Pegel können über SYSTEMBLÖCKE | DIGITALBLÖCKE | PEGEL vorgegeben werden.



Vorgabe der Logik-Pegel für Digitalbausteine

Der Wert für den *Schaltpegel* gibt dabei den Schwellwert an, ab dem ein digitaler Eingangswert als logisch 1 angesehen wird. Voreingestellt sind die Werte 0 für low, 5 für high und 2.5 für den Umschaltwert (entsprechend TTL).

Aktion

Dies sind Blöcke, die dem Benutzer einen interaktiven Eingriff gestatten (z. B. Schalter).

Kommunikation

Diese Gruppe umfasst Blöcke zur Kommunikation über DDE oder das TCP/IP-Protokoll.

Simulation

Hierunter fallen Blöcke, die die Simulationssteuerung selbst betreffen.

Senken

Diese Typklasse umfasst all jene Blöcke, die lediglich einen bzw. mehrere Eingänge besitzen. Dies sind in der Regel Blocktypen zur Visualisierung oder Weiterverarbeitung von Simulationsergebnissen (Beispiel: Oszillograph, File-Output). Erstere weisen neben dem Systemblock im Zeichenfenster zusätzlich noch das eigentliche *Anzeigefenster* auf, innerhalb dessen die Simulationsergebnisse dargestellt werden.[1] Beim Einfügen eines solchen Blocks wird dieses Anzeigefenster

[1] Auch andere Blocktypen (z. B. Fuzzy Controller) können ein Anzeigefenster besitzen.

zunächst in Symboldarstellung mit einem typspezifischen Icon dargestellt. Zur Simulation können alle Anzeigefenster über ANSICHT | ALLE ANZEIGEFENSTER ZEIGEN in Normalgröße dargestellt werden. Entsprechend können sie über ANSICHT | ALLE ANZEIGEFENSTER VERBERGEN jederzeit wieder in die Symboldarstellung zurückversetzt werden. Das Anzeigefenster weist jeweils den gleichen Titel auf wie der Systemblock selbst. Die Parametrierung von Ausgangsblöcken kann wahlweise durch einen Doppelklick auf den Systemblock oder innerhalb des Anzeigefensters erfolgen. Bei einigen der Ausgangsblöcke sind die Anzeigefenster zudem in ihrer Größe veränderbar (z. B. Oszillograph).

Sonstige

Diese Klasse enthält alle Blöcke, die nicht eindeutig einer der vorgenannten Gruppen zugeordnet werden können.

Hinweis: Die in den folgenden Abschnitten mit einem Sternchen (*) versehenen Blocktypen sind nur bei Erwerb der entsprechenden BORIS-Zusatzmodule bzw. Hardwarekomponenten und -treiber verfügbar!

Eingangsblöcke (Quellen)

Generator

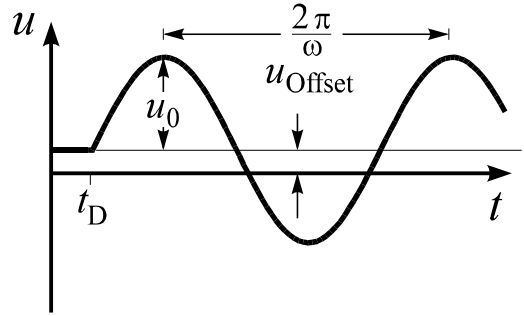
Typname: GENERATOR

Funktion: Dient zur Erzeugung von Testsignalen verschiedenster Art zur Anregung von Systemen. Der Generator weist verschiedene Betriebsarten auf, die in der Systemstruktur jeweils durch unterschiedliche Block-Bitmaps gekennzeichnet werden:

- Sinusgenerator

  In diesem Fall erzeugt der Generator ein sinusförmiges Ausgangssignal der Form

$$u(t) = \begin{cases} u_{\text{Offset}} & \textit{für } t < t_{\text{D}} \\ u_0 \sin(\omega(t - t_{\text{D}}) + \varphi) + u_{\text{Offset}} & \textit{sonst} \end{cases}$$

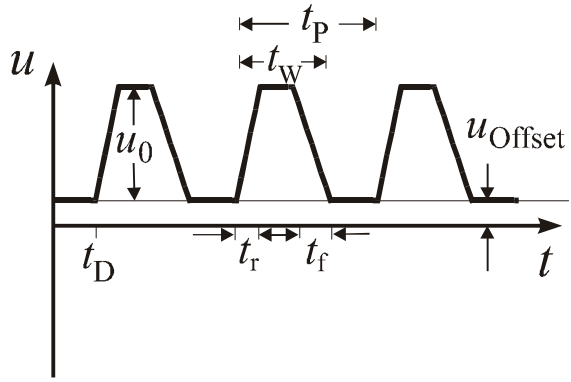


Parameter:

u_0 :	Amplitude
u_{Offset} :	Offset
t_{D} :	Verzugszeit
ω :	Kreisfrequenz
φ :	Phase (in Grad)

- Pulsgenerator

  In diesem Fall erhält man je nach Parameterwahl Rechteckimpulse, Dreieckimpulse oder auch Sägezahnimpulse:



Parameter:

t_{r} :	Anstiegszeit
t_{f} :	Abstiegszeit
t_{w} :	Pulsweite

t_p : Periodendauer

Restliche Parameter wie bei Sinusgenerator

Durch spezielle Wahl der Parameter lassen sich auch Einzelimpulse erzeugen. Wählt man beispielsweise Pulsweite und Periodendauer größer als die Simulationsdauer und An- bzw. Abstiegszeit zu null, so erhält man eine Sprungfunktion (dies ist die Voreinstellung!).

- Rauschgenerator

  In diesem Fall erzeugt der Generator zu jedem Simulationszeitpunkt eine gleichverteilte Zufallszahl aus dem Intervall $[-u_0 + u_{\text{Offset}},\ u_0 + u_{\text{Offset}}]$.

- *Programmierbarer Funktionsgenerator*

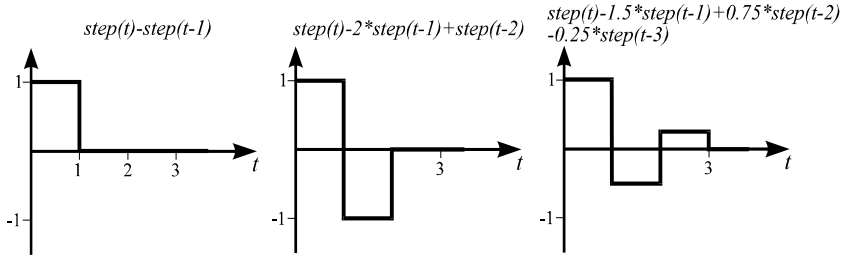
  Für spezielle Anwendungen kann der Generator vom Anwender über eine nahezu beliebige Funktion programmiert werden, die über einen Funktionsparser interpretiert wird. Diese Betriebsart ist allerdings relativ rechenzeitaufwendig. Der Parser erlaubt die Interpretation folgender Symbole bzw. Funktionen:

Syntax	**Funktion**
t	Unabhängige Variable (Zeit)
+	Plus (Addition)
-	Minus und monadisches Minus
*	Multiplikation
/	Division
^	Potenzoperator
( )	Klammern (max. Verschachtelungstiefe 20)
ln	natürlicher Logarithmus
log	Zehnerlogarithmus
sqr	Quadrat
sqrt	Wurzel
exp	Exponentialfunktion
sin	Sinus
cos	Cosinus
tan	Tangens
asin	Arcussinus
acos	Arcuscosinus
atan	Arcustangens

sinh	Sinus hyperbolicus
cosh	Cosinus hyperbolicus
tanh	Tangens hyperbolicus
abs	Absolutwert
deg	Umrechnung Radiant → Grad
rad	Umrechnung Grad → Radiant
int	ganzzahliger Anteil
frac	gebrochener Anteil
round	rundet auf ganze Zahl
sign	Signumfunktion
step	Sprungfunktion (Heavisidefunktion)

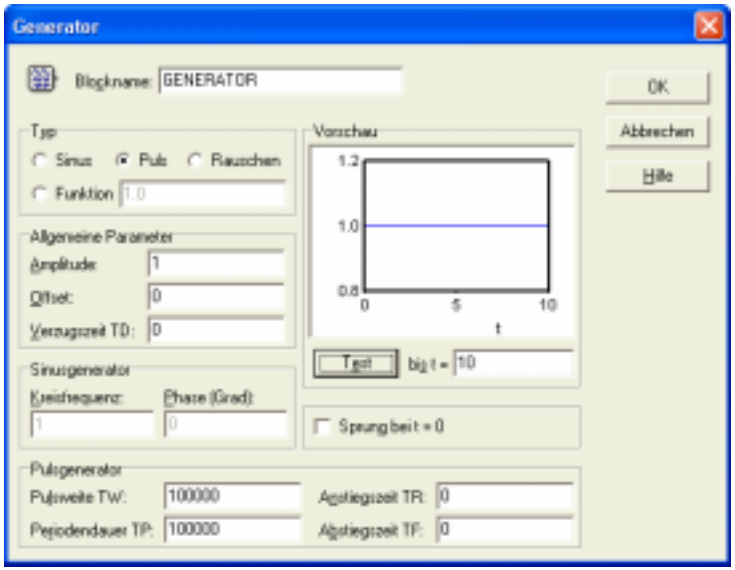
Der Funktionsstring darf maximal 255 Zeichen aufweisen. Groß- und Kleinschreibung werden nicht unterschieden.

Über den Funktionsgenerator lassen sich in Verbindung mit der Sprungfunktion z. B. auf einfache Weise nahezu beliebige Impulsformen realisieren. Die nachfolgenden Kurven zeigen einige Beispiele jeweils zusammen mit dem entsprechenden Funktionsaufruf.



Erzeugung diverser Impulsformen über die step-Funktion

Parameter-dialog:



Die Schaltergruppe *Typ* dient zur Wahl der Betriebsart. Die aktuellen Einstellungen können über die Schaltfläche *Test* jederzeit ausgetestet werden.

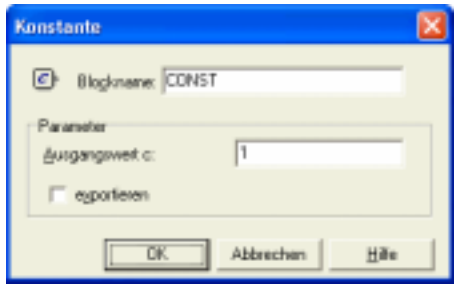
Ist das Schaltfeld *Sprung bei t=0* markiert, so wird die Ausgangsgröße des Generators unabhängig von den gewählten Einstellungen zum Simulationsbeginn auf null gesetzt. Die eigentliche Ausgangsgröße wird in diesem Fall erst beim ersten Simulationsschritt, d. h. nach einer Simulationsschrittweite ΔT aufgeschaltet.

Konstante

Typname: CONST

Funktion: Gibt einen konstanten Wert aus.

Parameter-dialog:

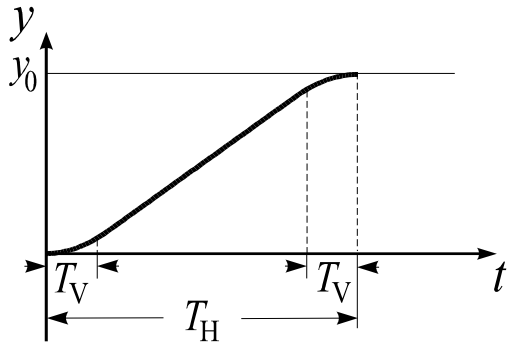


Ausgangswert gibt den ausgegebenen Wert an.

Fahrkurve

Typname: FAHRKURVE

Funktion: Der Eingangsblock erzeugt ein gleichmäßig von null auf den Endwert ansteigendes Signal, wie es beispielsweise zum Hochfahren von Motoren benutzt wird. Nachfolgende Grafik zeigt den prinzipiellen Verlauf des Signals.



Es gilt für den Verlauf von $y(t)$:

$$y(t) = \frac{y_0 t^2}{2(T_H - T_V)T_V}\Bigg|_{t=0}^{t=T_V} + \frac{y_0}{T_H - T_V}(t - T_V)\Bigg|_{t=T_V}^{t=T_H - T_V} + \ldots$$
$$\ldots + \frac{y_0 T_V}{2(T_H - T_V)}\left(1 - \frac{(t - T_H)^2}{T_V^2}\right)\Bigg|_{t=T_H - T_V}^{t=T_H}$$

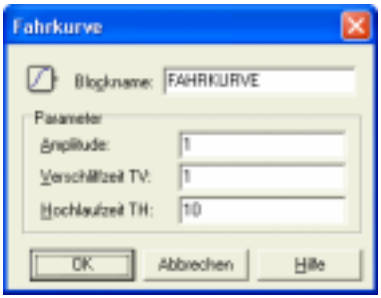
Darin ist:

y_0: Amplitude des Signals

T_V: Verschliffzeit

T_H: Hochlaufzeit

Parameter-dialog:

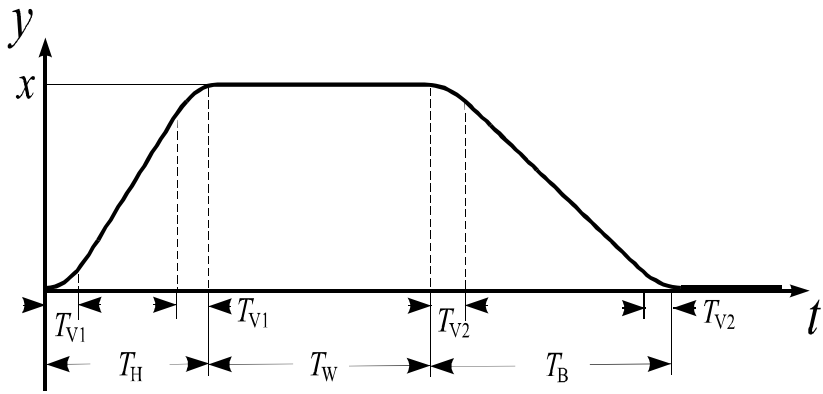


Parameter-grenzen: $T_H \geq 2T_V$

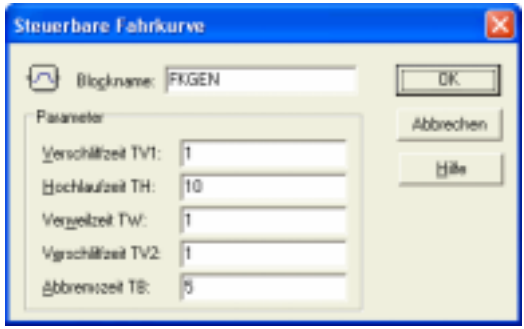
Steuerbare Fahrkurve (Fahrkurvengenerator)

Typname: FKGEN

Funktion: Dieser Blocktyp stellt eine Erweiterung des vorangegangenen Blocktyps FAHRKURVE dar. Das generierte Ausgangssignal steigt zunächst innerhalb einer Hochlaufzeit T_H auf den am Blockeingang x anliegenden (Soll-) Wert an und fällt dann nach einer Verweildauer T_W innerhalb einer Abbremszeit T_B wieder auf null zurück. Über eine positive Flanke am Reset-Eingang R kann der Ausgang jederzeit zurückgesetzt werden. Nachfolgende Grafik zeigt den prinzipiellen Verlauf des Signals.



Parameter-dialog:



Parameter-grenzen: $T_H \geq 2T_{V1}, T_B \geq 2T_{V2}$

Steuerbarer Sinusgenerator (VCO)

Typname: VCO

Funktion: Dieser Block stellt einen Sinusgenerator dar, dessen Frequenz über die Eingangsgröße variiert werden kann. Die Variation kann linear oder exponentiell erfolgen.

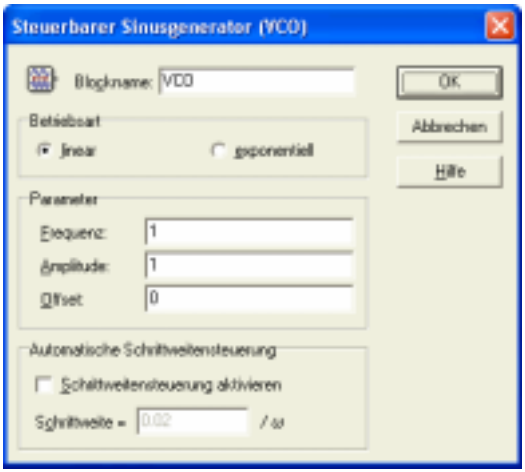
Die Grundwerte entsprechen den Einstellungen des Blocktyps *Generator* in der Betriebsart als Sinusgenerator. Ist ω_0 die Grundfrequenz des VCO, so ergibt

sich die aktuelle Frequenz ω in Abhängigkeit von der Eingangsgröße x wie folgt:

Betriebsart *linear*: $\omega = x\omega_0$

Betriebsart *exponentiell*: $\omega = 10^x \omega_0$

Parameter-dialog:



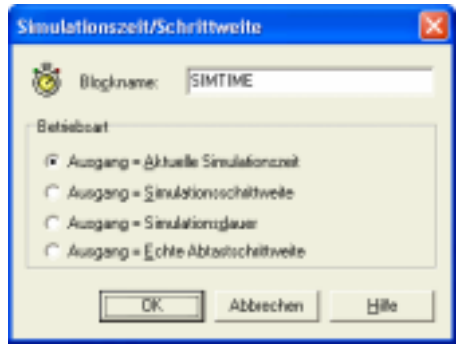
Die Einstellungen in der Gruppenbox *Automatische Schrittweitensteuerung* können im Batch-Betrieb für eine automatische Anpassung der Simulationsschrittweite an die Frequenz benutzt werden. Hinweise dazu finden Sie im Abschnitt *Simulationen im Batch-Betrieb*.

Simulationszeit/Schrittweite

Typname: SIMTIME

Funktion: Dieser Block gibt wahlweise die verstrichene Simulationszeit, die Simulationsschrittweite oder die Simulationsgesamtdauer aus.

Parameter-dialog:

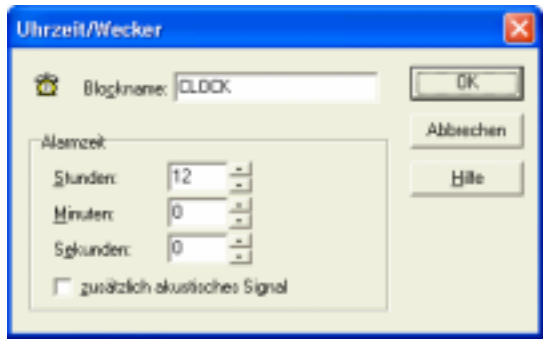


Uhrzeit/Wecker

Typname: CLOCK

Funktion: Echtzeituhr mit Alarmeinrichtung. An den Ausgängen H, M und S wird die aktuelle Zeit in Stunden, Minuten und Sekunden ausgegeben. Bei Erreichen der voreingestellten Alarmzeit erhält der Alarmausgang A für einen Simulationsschritt High-Pegel. Auf Wunsch kann zusätzlich ein Signalton ausgegeben werden.

Parameter-dialog:



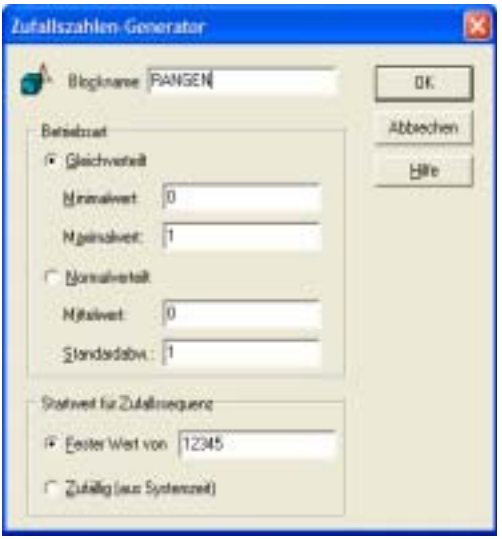
Zufallszahlen-Generator

Typname: RANGEN

Funktion: Dieser Blocktyp erlaubt die Erzeugung gleich- oder normalverteilter Zufallszahlen. Bei einer Gleichverteilung müssen untere und obere Grenze der erzeugten Zufallszahlen vorgegeben werden, bei einer Normalverteilung Mittelwert und Standardabweichung.

Der Startwert für die Zufallssequenz kann vorgegeben werden; für einen festen Startwert ergibt sich dann bei jedem Simulationslauf eine *identische* Sequenz von Zufallszahlen. Soll sich bei jedem Lauf eine andere Sequenz ergeben, kann der Startwert alternativ aus der Systemzeit abgeleitet werden.

Parameterdialog:



Datei

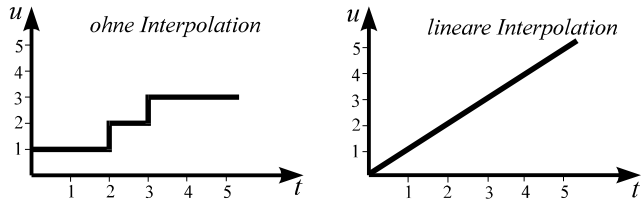
Typname: FILEINPUT

Funktion: Ermöglicht das Einlesen eines Signals $u(t)$ in Form von Wertepaaren (t_i, u_i) aus einer Datei vom Typ SIM. Die Zeitpunkte t_i können beliebig (d. h. nicht zwangsläufig äquidistant), müssen aber in aufsteigender Reihenfolge sortiert sein. Zwischen den eingelesenen Werten kann linear interpoliert werden. Andernfalls wird der letzte gültige Wert jeweils beibehalten, bis ein neuer Wert in der Datei auftritt. In diesem Fall ergibt sich ein stufenförmiger Verlauf der Eingangsgröße.

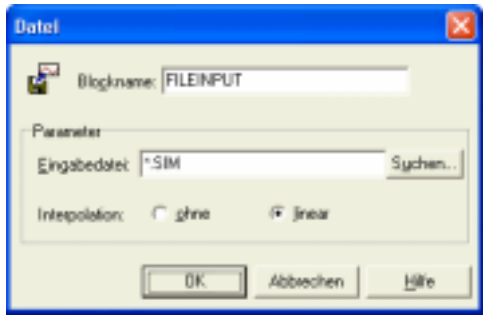
Beispiel: Die eingelesene Datei enthalte die drei Wertepaare

1	1
2	2
3	3.

Für eine Simulation bis zum Zeitpunkt T_{Simu} = 5 ergeben sich dann folgende Signalverläufe:



Parameter-dialog:



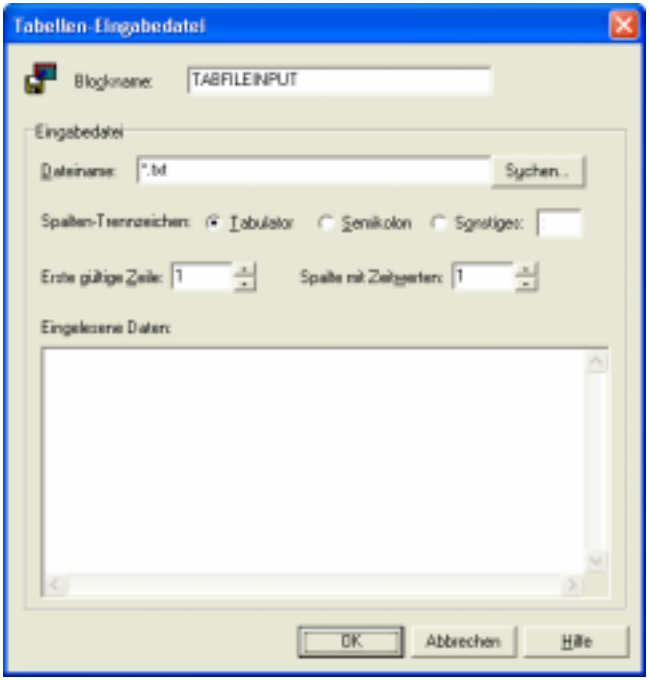
Wird für *Eingabedatei* keine Extension angegeben, wird die Extension SIM benutzt. Über *Suchen* kann ein Dateieingabedialog angefordert werden.

Tabellen-Eingabedatei

Typname: TABFILEINPUT

Funktion: Dieser Block ermöglicht das Einlesen von Zeitverläufen aus einer strukturierten ASCII-Datei, wobei jede Zeile der Datei einen Zeitwert und die zugehörigen Amplitudenwerte enthält. Am Dateianfang können sich beliebig viele Kommentarzeilen (z. B. Spaltenüberschriften) befinden. Der Spaltenseparator ist frei wählbar.

Parameterdialog:



Im Feld *Erste gültige Zeile* ist die erste mit zu interpretierenden Zahlenwerten gefüllte Zeile anzugeben. Das Feld *Spalte mit Zeitwerten* gibt an, welche der Spalten den Zeitwert enthält. Die Zeitwerte müssen in aufsteigender Folge sortiert sein; der Abstand zwischen zwei Werten muss jedoch *nicht* der Simulationsschrittweite entsprechen; ggf. wird linear interpoliert. Wird für *Dateiname* keine Extension angegeben, wird die Extension TXT benutzt. Über *Suchen* kann ein Dateieingabedialog angefordert werden.

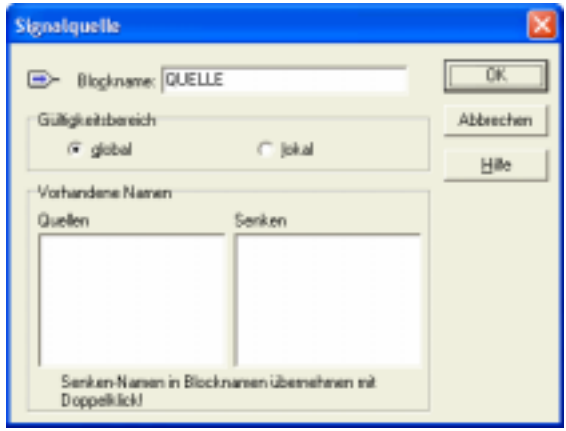
Signalquelle

Typname: QUELLE

Funktion: Dieser Block dient zusammen mit dem Ausgangsblocktyp *Signalsenke* zur Realisierung "drahtloser" Verbindungen zwischen Blöcken. Dabei "versendet" eine Signalsenke ihr Eingangssignal unter einem bestimmten Namen (dem Namen des Blocks). Dieses Signal kann dann an beliebigen - auch mehreren - Stellen in der Systemstruktur von einer Signalquelle mit gleichem Namen wieder empfangen werden. Groß- und Kleinschreibung der Blocknamen wird dabei nicht unterschieden. Signalquellen können *lokale* oder *globale* Gültigkeit haben. Während sie im ersten Fall nur innerhalb der zugehörigen System- bzw. Superblockdatei bekannt sind, gelten sie im globalen Fall auch über die Dateigrenzen hinaus. Der Einsatz beider Blocktypen empfiehlt sich insbesondere bei komplexen, stark vermaschten Systemstrukturen, um die Anzahl der sichtbaren Verbindungen gering zu halten.

Das Quellen/Senken-Konzept

Parameterdialog:



In der linken Listbox des Dialogs sind alle bereits vorhandenen Quellen in alphabetischer Reihenfolge aufgelistet, in der rechten Listbox alle Senken. Durch Doppelklick auf einen Senken-Namen in der rechten Listbox kann dieser direkt als Quellen-Name übernommen werden.

Dynamische Blöcke

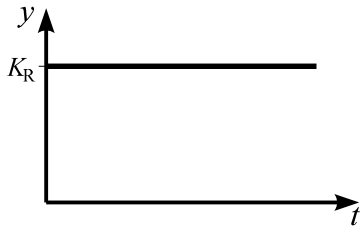
P-Glied

Typname: P

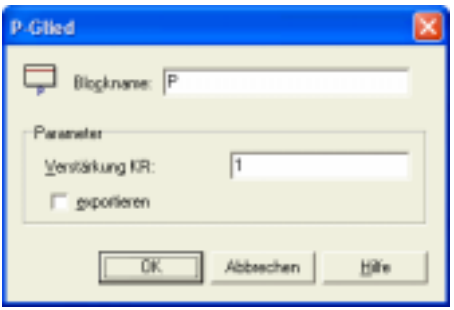
Funktion: Das Proportionalglied (P-Glied) erzeugt aus der Eingangsgröße $x(t)$ eine Ausgangsgröße $y(t)$ gemäß der Beziehung $y(t) = K_R x(t)$. Es besitzt somit die Übertragungsfunktion

$$G(s) = K_R$$

und die nachfolgende Sprungantwort:



Parameter-dialog:



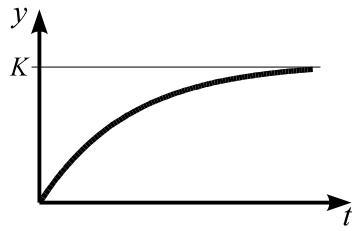
PT_1-Glied

Typname: PT1

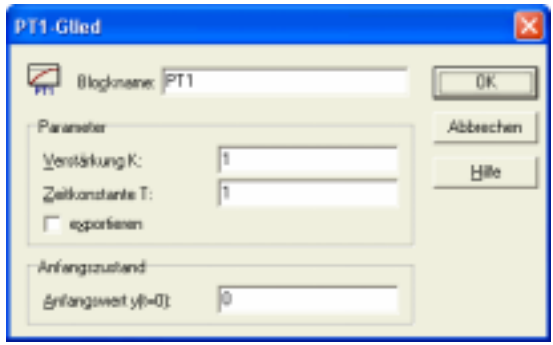
Funktion: Verzögerungsglied 1. Ordnung mit der Verstärkung K und der Zeitkonstanten T. Das PT_1-Glied besitzt demnach die Übertragungsfunktion

$$G(s) = \frac{K}{1+Ts}$$

und die nachfolgende Sprungantwort:



Parameter-dialog:



Für die Simulation muss der Anfangswert $y(t=0)$ vorgegeben werden.

Parameter-grenzen: $T > 0$

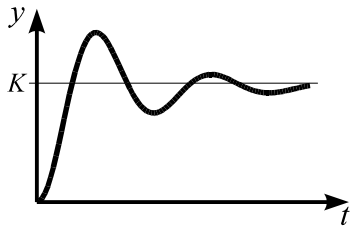
PT_2-Glied (schwingfähig)

Typname: PT2

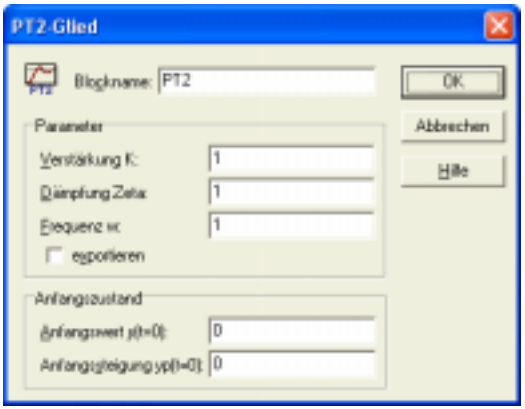
Funktion: Schwingfähiges Verzögerungsglied 2. Ordnung mit der Verstärkung K, der Eigenfrequenz ω und der Dämpfung ζ. Das System besitzt die Übertragungsfunktion

$$G(s) = \frac{K}{\left(\frac{s}{\omega}\right)^2 + 2\frac{\zeta}{\omega}s + 1}$$

und nachfolgende Sprungantwort:



Parameterdialog:



Für die Simulation müssen der Anfangswert $y(t=0)$ und die Anfangssteigung $\dot{y}(t=0)$ vorgegeben werden.

Parameter-grenzen: $\zeta \geq 0,\ \omega > 0$

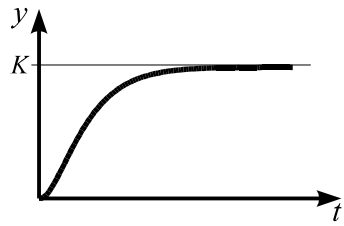
PT_1T_2-Glied

Typname: PT1T2

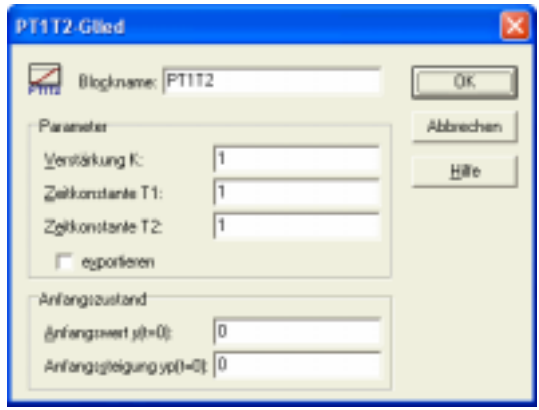
Funktion: Nicht schwingfähiges Verzögerungsglied 2. Ordnung mit der Verstärkung K und den beiden Zeitkonstanten T_1 und T_2. Die Übertragungsfunktion lautet

$$G(s) = \frac{K}{(1+T_1 s)(1+T_2 s)}.$$

Das System besitzt nachfolgende Sprungantwort:



Parameter-dialog:



Für die Simulation müssen der Anfangswert $y(t=0)$ und die Anfangssteigung $\dot{y}(t=0)$ vorgegeben werden.

Parameter-grenzen: $T_1, T_2 > 0$

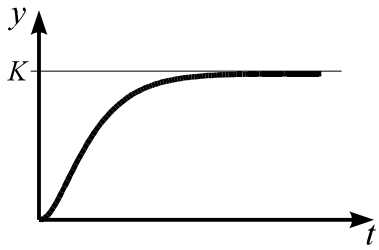
PT_n-Glied

Typname: PTN

Funktion: Verzögerungsglied *n*-ter Ordnung mit der Verstärkung *K* und *n* gleichen Zeitkonstanten *T*. Die zugehörige Übertragungsfunktion lautet

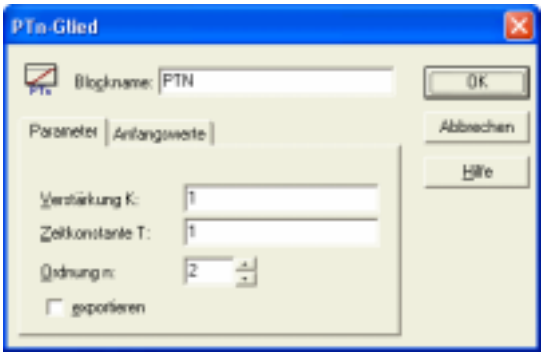
$$G(s) = \frac{K}{(1+Ts)^n}.$$

Das System besitzt die folgende Sprungantwort:



Bei konstantem *T* verläuft die Sprungantwort mit steigendem Wert für *n* zunehmend flacher.

Parameter-dialog:



Parameter-grenzen: $T > 0, \quad 1 \le n \le 8$

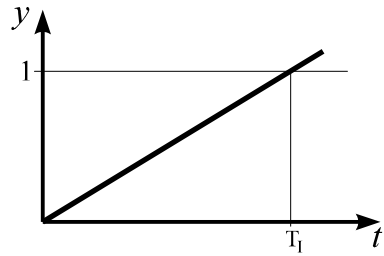
Begrenzter Integrierer

Typname: I

Funktion: Begrenzter Integrierer mit der Integrations-Zeitkonstanten T_I. Innerhalb des linearen Arbeitsbereiches besitzt das System die Übertragungsfunktion

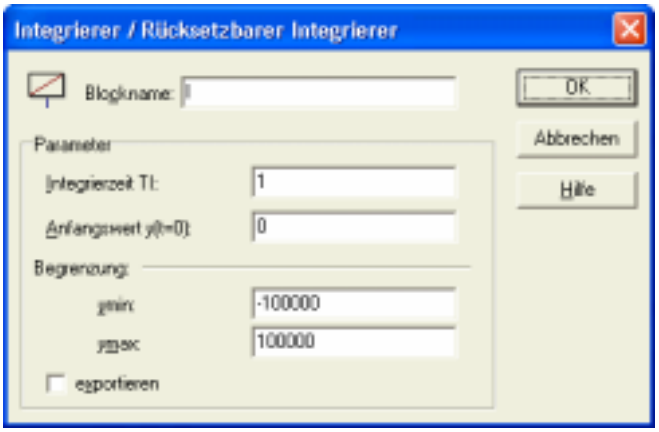
$$G(s) = \frac{1}{T_I s}$$

und nachfolgende Sprungantwort:



Die Ausgangsgröße $y(t)$ des Integrierers wird durch eine Anti-Windup-Halt-Maßnahme auf den Wertebereich $y_{min} \le y(t) \le y_{max}$ begrenzt. Diese hält die Integration an, solange die Ausgangsgröße an der oberen Begrenzung y_{max} ist und die Eingangsgröße des Integrierers positiv bzw. die Ausgangsgröße an der unteren Begrenzung y_{min} und die Eingangsgröße negativ ist. Bei Vorzeichenwechsel der Eingangsgröße löst sich die Ausgangsgröße dann jeweils unmittelbar von der Begrenzung.

Parameter-dialog:



Zur Simulation muss der Anfangswert $y(t=0)$ vorgegeben werden.

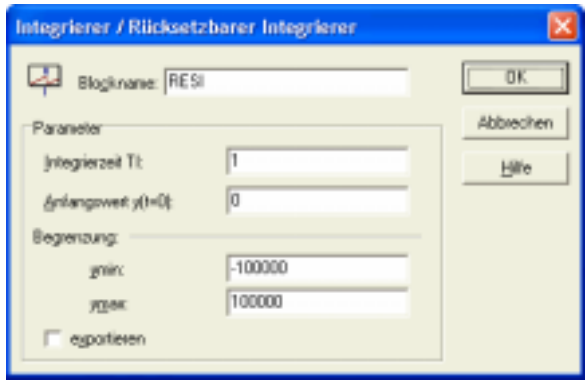
Parameter-grenzen: $T_I > 0$

Rücksetzbarer Integrierer

Typname: RESI

Funktion: Der Block entspricht dem begrenzten Integrierer, die Ausgangsgröße kann während der Simulation jedoch durch eine positive Flanke am Steuereingang R jederzeit auf den Anfangswert zurückgesetzt werden.

Parameter-dialog:



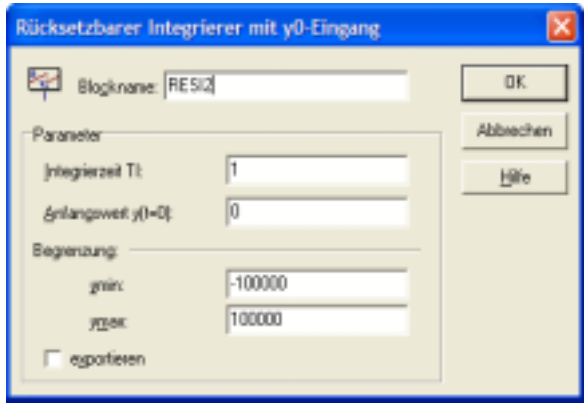
Parameter-grenzen: $T_I > 0$

Rücksetzbarer Integrierer mit y_0-Eingang

Typname: RESI2

Funktion: Der Block entspricht dem rücksetzbaren Integrierer, wobei die Ausgangsgröße durch eine positive Flanke am Steuereingang R jedoch auf den am Eingang y0 anliegenden Wert zurückgesetzt wird.

Parameter-dialog:



Parameter-grenzen: $T_I > 0$

Differenzierer

Typname: D

Funktion: Idealer Differenzierer mit der Differentiations-Zeitkonstanten T_D , der Übertragungsfunktion

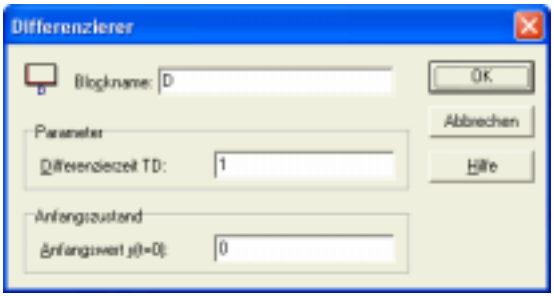
$$G(s) = T_D s$$

und nachfolgender Sprungantwort:

Die Ausgangsgröße nimmt zum Zeitpunkt $t = 0$ theoretisch den Wert ∞ an. In der Simulation wird der Ausgangswert bei einem Eingangssprung der Höhe Δx und der Simulationsschrittweite ΔT begrenzt auf

$$y_{max} = \frac{\Delta x}{\Delta T}.$$

Parameter-dialog:



Zur Simulation muss der Anfangswert $y(t=0)$ des Differenzierers vorgegeben werden.

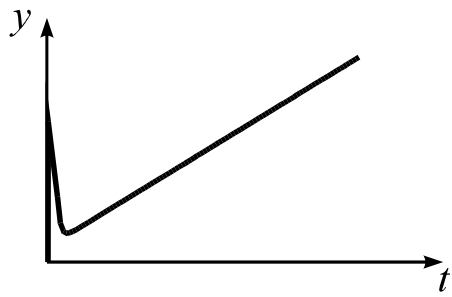
PID-Regler

Typname: PID

Funktion: PID-Regler mit der Verstärkung K_R, der Nachstellzeit T_N und der Vorhaltezeit T_V. Der D-Anteil des Reglers ist mit einem zusätzlichen PT_1-Glied versehen (verzögerte Differentiation). Die Ausgangsgröße $y(t)$ wird durch eine Anti-Windup-Halt-Maßnahme auf den Wertebereich $y_{min} \le y(t) \le y_{max}$ beschränkt. Im linearen Arbeitsbereich weist der PID-Regler die Übertragungsfunktion

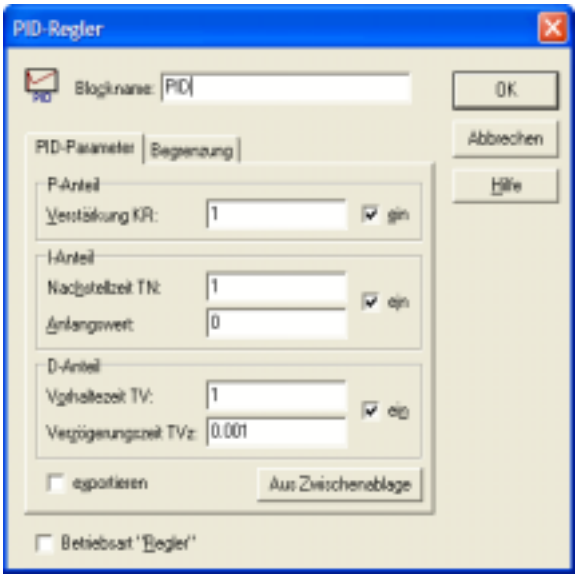
$$G(s) = K_R\left(1 + \frac{1}{T_N s} + \frac{T_V s}{1 + T_{Vz} s}\right)$$

und folgende Sprungantwort auf:



P-, I- und D-Anteil sind getrennt voneinander zu- und abschaltbar. Ebenso kann die Begrenzung zu- oder abgeschaltet werden. Für die Simulation muss der Anfangswert des Integrierers vorgegeben werden.

Parameter-dialog:



In der Betriebsart *Regler* weist der Block zwei Eingangsgrößen (Führungsgröße *w* und Rückführgröße *r*) auf; die Regeldifferenz wird in diesem Fall blockintern aus der Differenz dieser beiden Größen berechnet.

Parameter-grenzen: $T_N, T_{Vz} > 0, \quad T_V \geq 0$

Adaptiver PID-Regler mit steuerbarer Begrenzung

Typname: ADAPID

Funktion: Der Block entspricht in seiner Funktionsweise dem zuvor beschriebenen PID-Regler, die Parameter K_R, T_N und T_V können jedoch während der Simulation über die Steuereingänge P, I und D modifiziert werden. Die Modifikation kann multiplikativ oder additiv erfolgen. Die aktuellen Werte für die Parameter ergeben sich aus den über den Parameterdialog eingestellten Grundwerten K_{R0}, T_{N0} und T_{V0} wie folgt:

Betriebsart *multiplikativ*:

$$K_R = K_{R0} x_P(t)$$
$$T_N = T_{N0} x_I(t)$$
$$T_V = T_{V0} x_D(t)$$

Betriebsart *additiv*:

$$K_R = K_{R0} + x_P(t)$$
$$T_N = T_{N0} + x_I(t)$$
$$T_V = T_{V0} + x_D(t)$$

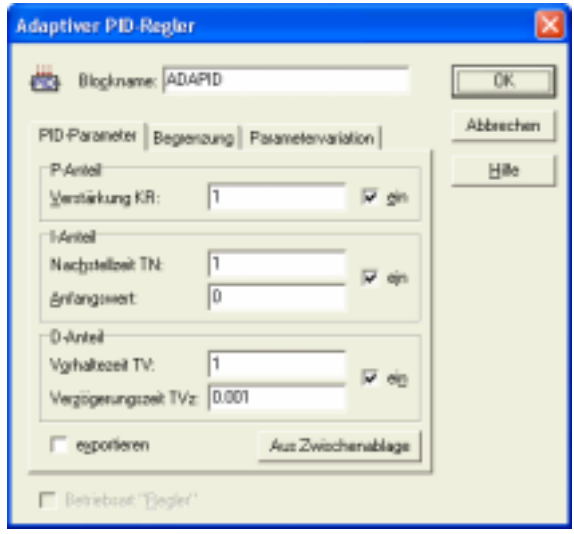
$x_P(t), x_I(t)$ und $x_D(t)$ sind die an den Steuereingängen P, I und D anliegenden Signale. In der Betriebsart multiplikativ werden offene Steuereingänge zu eins, in der Betriebsart additiv zu null gesetzt.

Über den Steuereingang S kann darüber hinaus die Begrenzung des Reglers von außen beeinflusst werden. Ist der Steuereingang offen, so arbeitet der Regler mit der festen Begrenzung der Ausgangsgröße auf den Bereich $y_{min} \le y(t) \le y_{max}$. Ist der Steuereingang angeschlossen, so ergibt sich die tatsächliche Begrenzung durch Multiplikation der über den Dialog eingestellten Werte mit dem am Steuereingang anliegenden Signal $x_S(t)$:

$$y_{min,\,akt}(t) = y_{min} \cdot x_S(t)$$
$$y_{max,\,akt}(t) = y_{max} \cdot x_S(t)$$

Voraussetzung ist natürlich jeweils, dass die Begrenzung auch aktiviert wurde!

Parameterdialog:



Parameter-grenzen: T_N, $T_{Vz} > 0$, $T_V \geq 0$

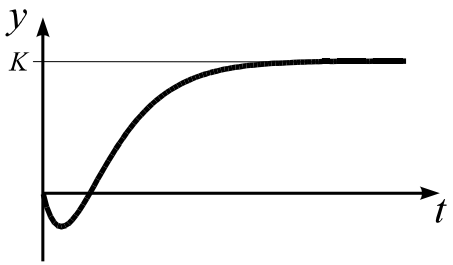
Allpass Typ I

Typname: ALLPASS_1

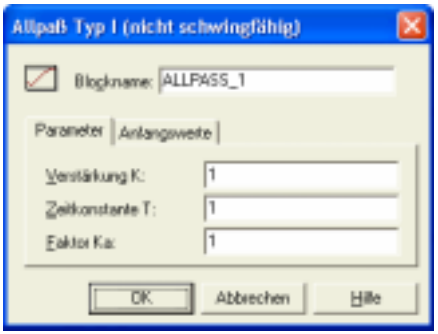
Funktion: Allpasssystem 2. Ordnung (nicht schwingfähig) mit der Übertragungsfunktion

$$G(s) = K\frac{1 - K_a Ts}{(1 + K_a Ts)(1 + Ts)}$$

und nachfolgender Sprungantwort:



Parameter-dialog:



Die Anfangswerte des Systems werden für die Simulation zu null gesetzt.

Parameter-grenzen: T, $K_a > 0$

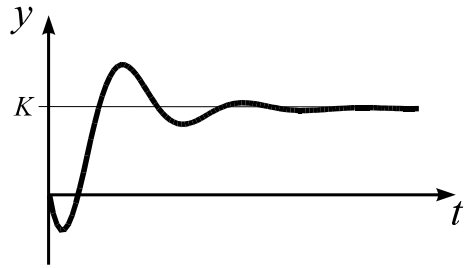
Allpass Typ II

Typname: ALLPASS_2

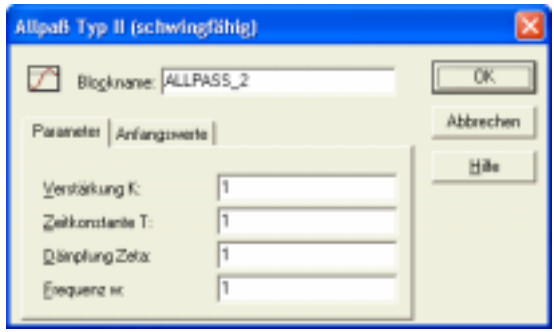
Funktion: Allpasssystem 2. Ordnung (schwingfähig) mit der Übertragungsfunktion

$$G(s) = K\frac{1-Ts}{\left(\frac{s}{\omega}\right)^2 + 2\frac{\zeta}{\omega}s + 1}$$

und der nachfolgenden Sprungantwort:



Parameter-dialog:



Die Anfangswerte des Systems werden für die Simulation zu null gesetzt.

Parameter-grenzen: $T,\ \omega > 0,\ \zeta \geq 0$

Totzeitglied

Typname: TOTZEIT

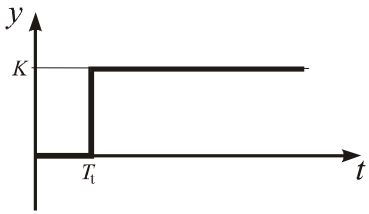
Funktion: Das Totzeitglied verzögert die Eingangsgröße $x(t)$ um die Totzeit T_t gemäß

$$y(t) = \begin{cases} y_0 & \text{für } t < T_t \\ K\,x(t - T_t) & \text{für } t \geq T_t \end{cases}$$

und besitzt die Übertragungsfunktion

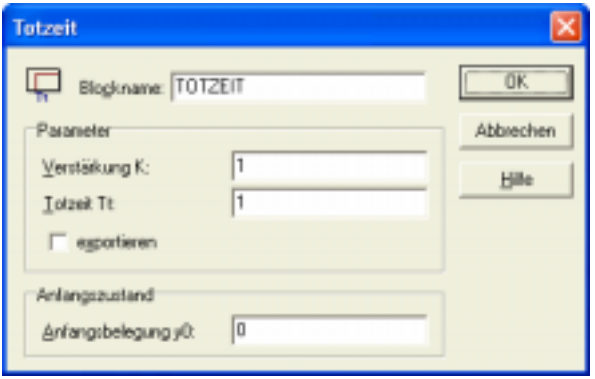
$$G(s) = Ke^{-T_t s}$$

sowie folgende Sprungantwort:



Der interne Speicher des Totzeitgliedes kann beliebig viele Elemente aufnehmen. Die Totzeit sollte möglichst ein ganzzahliges Vielfaches der Simulationsschrittweite ΔT betragen.

Parameterdialog:



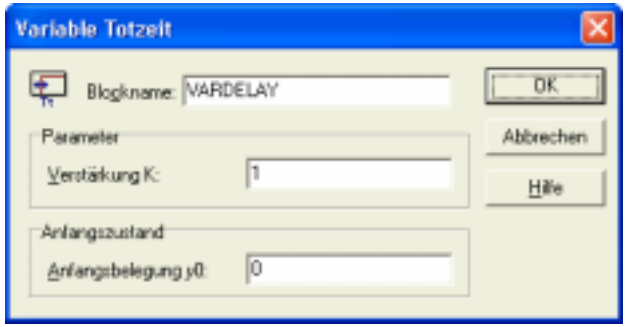
Parameter-grenzen: $T_t \geq 0$

Variable Totzeit

Typname: VARDELAY

Funktion: Dieser Block realisiert ein Totzeitglied mit variabler Totzeit und kann daher z. B. zur Modellierung von Systemen mit variabler Laufzeit (z. B. in der Verfahrenstechnik) benutzt werden. Das am Eingang E anliegende Signal wird um die am Eingang Tt anliegende Totzeit verzögert (siehe auch Blocktyp TOTZEIT). Der interne Speicher des Totzeitgliedes kann beliebig viele Elemente aufnehmen.

Parameter-dialog:



Parameter-grenzen: Der Wert für die Totzeit wird intern nach unten automatisch auf den Wert ΔT für die Simulationsschrittweite begrenzt.

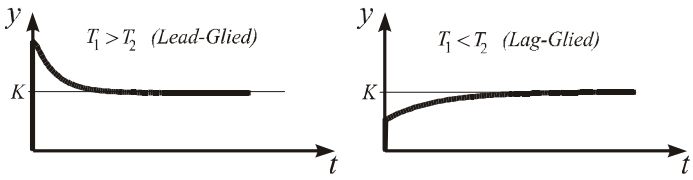
Lead/Lag-Glied

Typname: LEADLAG

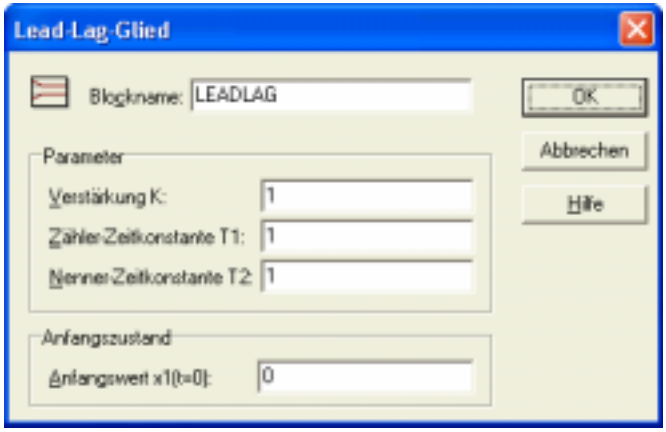
Funktion: Rationales Glied 1. Ordnung mit der Übertragungsfunktion

$$G(s) = K\frac{1+T_1 s}{1+T_2 s}$$

und den folgenden Sprungantworten:



Parameter-dialog:



Für die Simulation muss der Anfangswert $x_1(t=0)$ vorgegeben werden.

Parameter-grenzen: $T_1 \geq 0, \quad T_2 > 0$

Vorhalteglied (DT_1-Glied)

Typname: DT1

Funktion: Rationales Glied 1. Ordnung (verzögerter Differenzierer) mit der Übertragungsfunktion

$$G(s) = \frac{T_D s}{1 + T_1 s}$$

und der folgenden Sprungantwort:

Parameter-dialog:

Parameter-grenzen: T_D, $T_1 > 0$

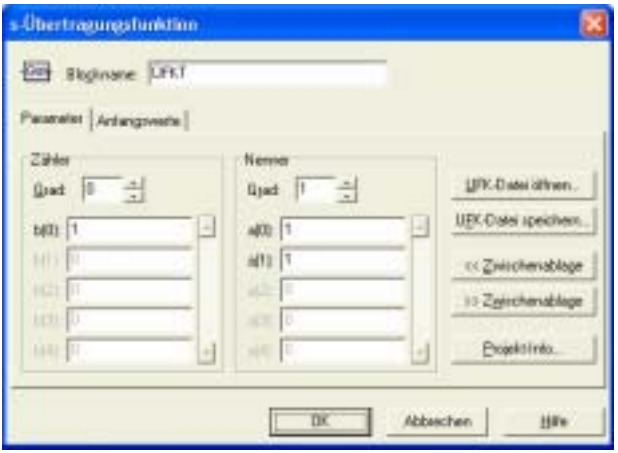
Übertragungsfunktion

Typname: ÜFKT

Funktion: Frei parametrierbare *s*-Übertragungsfunktion der Form

$$G(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0}, \quad m \le n \ .$$

Parameter-dialog:



Die Übertragungsfunktion kann wahlweise über Tastatur eingegeben oder aus der Zwischenablage bzw. einer UFK-Datei geholt werden.

Parameter-grenzen: $m \le n,\ n \le 8$

$a_n \neq 0$

Dgl.-System

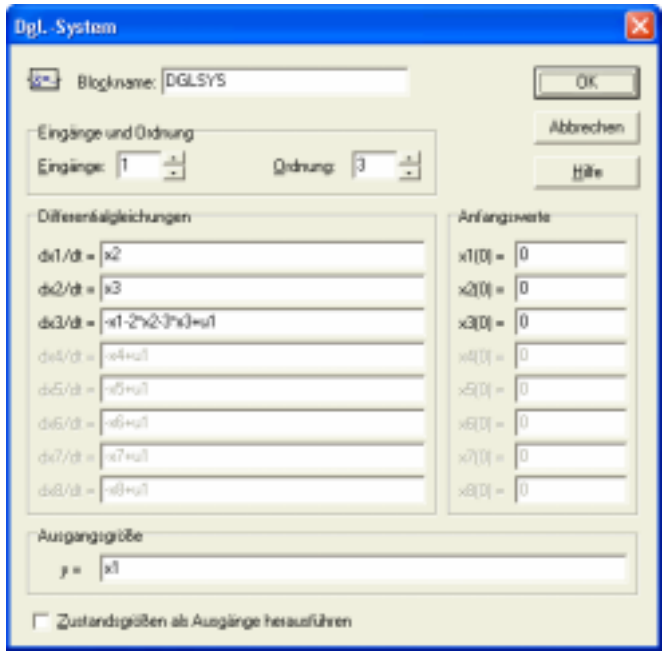
Typname: DGLSYS

Funktion: Frei parametrierbares Dgl.-System der Form

$$\begin{aligned} \dot{\underline{x}} &= F(\underline{x}, \underline{u}) \\ y &= g(\underline{x}, \underline{u}) \end{aligned} .$$

Das Dgl.-System kann sowohl linear als auch nichtlinear sein und wird über einen Funktionsparser interpretiert. $\underline{x}(t)$ ist der Zustandsvektor des Systems (maximale Ordnung: 8), $\underline{u}(t)$ der Eingangsvektor und $y(t)$ die Ausgangsgröße. Der Anfangswertvektor $\underline{x}(t = 0)$ kann ebenfalls frei vorgegeben werden.

Parameter-dialog:



Die Zustandsgrößen werden mit *x1*, *x2*, ... bezeichnet, die Eingangsgrößen mit *u1*, *u2*, ... Obiger Dialog enthält somit das - in diesem Fall lineare - Dgl.-System

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\dot{x}_3 = -x_1 - 2x_2 - 3x_3 + u_1$$

$$y = x_1$$

mit den Anfangswerten $\underline{x}(0) = \underline{0}$.

MIMO-Zustandsraummodell

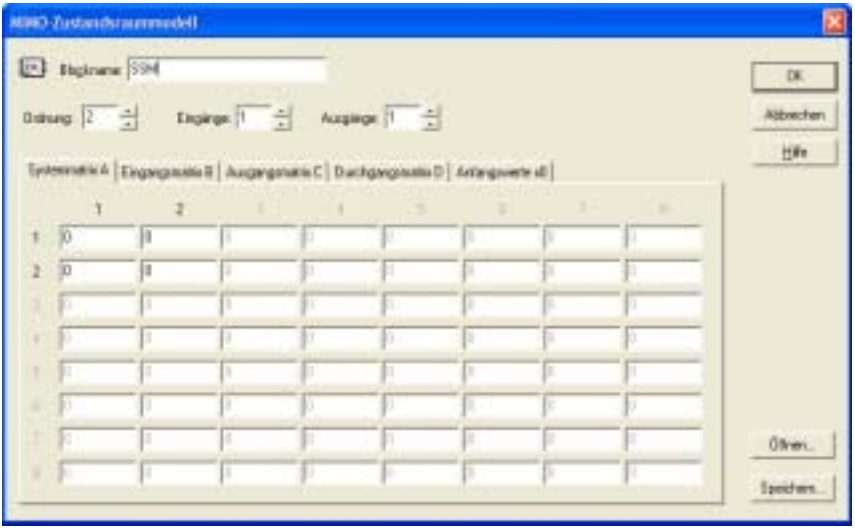
Typname: SSM

Funktion: Lineares Mehrgrößen-Zustandsraummodell der Form

$$\dot{\underline{x}} = \underline{A} \cdot \underline{x} + \underline{B} \cdot \underline{u}$$
$$\underline{y} = \underline{C} \cdot \underline{x} + \underline{D} \cdot \underline{u} \quad .$$

$\underline{x}(t)$ ist der Zustandsvektor des Systems (maximale Ordnung: 8), $\underline{u}(t)$ der Eingangsvektor und $\underline{y}(t)$ der Ausgangsvektor. Der Anfangswertvektor $\underline{x}(t=0)$ kann ebenfalls frei vorgegeben werden. $\underline{A}$ ist die Systemmatrix des Modells, $\underline{B}$ die Eingangsmatrix, $\underline{C}$ die Ausgangsmatrix und $\underline{D}$ die Durchgangsmatrix.

Parameterdialog:



Blockliste

Typname: BLOCKLIST

Funktion: Dieser Block enthält eine Liste linearer Standardglieder, die als Reihenschaltung interpretiert wird. Die Liste kann mit einem Passwortschutz versehen werden, sodass die interne Struktur der Liste nur nach Eingabe des korrekten Passworts eingesehen und modifiziert werden kann. Einzelheiten zur Konfigurierung der Blockliste entnehmen Sie bitte dem Kapitel 2 *Grundlagen*.

Parameter-dialog:

Einheitsverzögerung

Typname: UNITDELAY

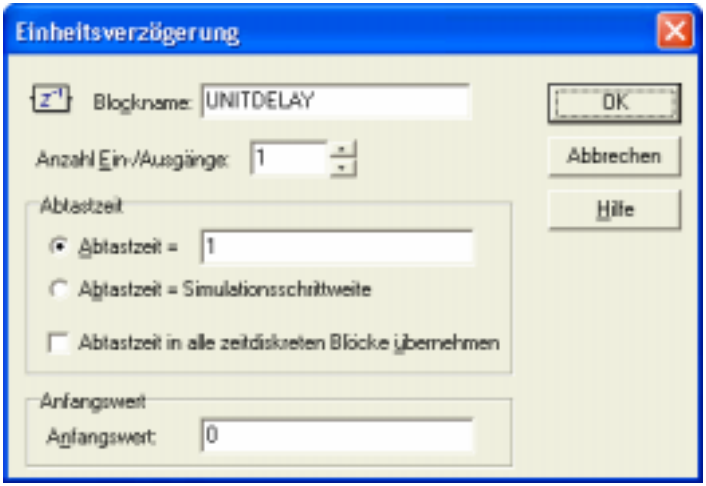
Funktion: Dieser Block realisiert ein Abtast-/Halteglied mit einer Verzögerung um eine Abtastperiode *T*. Er besitzt somit die z-Übertragungsfunktion

$$G(z) = \frac{1}{z}.$$

Die Abtastzeit T und der Anfangswert zu Beginn der Simulation sind frei wählbar. Der Block kann bis zu 50 Ein- und Ausgänge besitzen.

Dieser Blocktyp kann sinnvollerweise auch zur Verhinderung *algebraischer Schleifen* eingesetzt werden, um innerhalb einer Rückführung eine Verzögerung von genau einem Takt zu erreichen. Damit in solchen Fällen bei einer Änderung der Simulationsschrittweite nicht jedesmal die Abtastzeit angepasst werden muss, kann diese über den Parameterdialog direkt an die Abtastzeit angekoppelt werden.

Parameterdialog:



Ist die Option *Abtastzeit in alle zeitdiskreten Blöcke übernehmen* aktiviert, so wird die Abtastzeit des Blocks nach dem Verlassen des Dialogs automatisch in alle anderen zeitdiskreten Blöcke (d. h. Blöcke vom Typ UNITDELAY oder ZÜFKT) übernommen.

Die Abtastzeit sollte ein ganzzahliges Vielfaches der Simulationsschrittweite betragen, andernfalls erfolgt zu Beginn der Simulation eine Warnmeldung.

Parametergrenzen: $T > 0$

z-Übertragungsfunktion

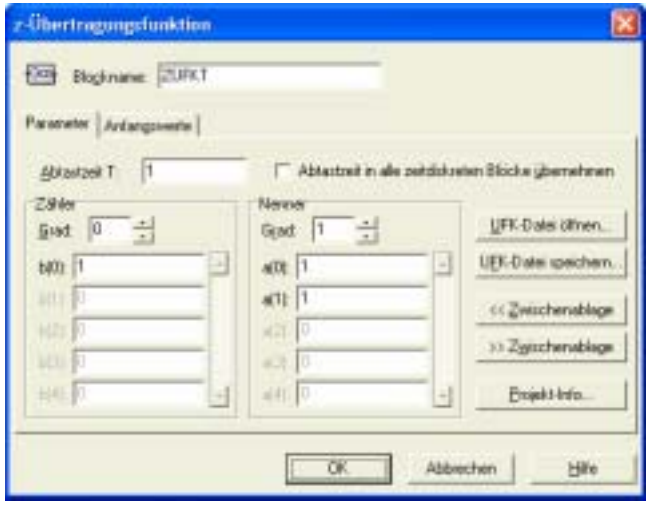
Typname: ZÜFKT

Funktion: Frei parametrierbare *z*-Übertragungsfunktion der Form

$$H(z) = \frac{b_m z^m + b_{m-1} z^{m-1} + \ldots + b_1 z + b_0}{a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0}, \quad m \le n \; .$$

Die Abtastzeit *T* ist frei wählbar.

Parameter-dialog:



Die Übertragungsfunktion kann wahlweise über Tastatur eingegeben oder aus der Zwischenablage bzw. einer UFK-Datei geholt werden. Das Dateiformat entspricht dem der *s*-Übertragungsfunktion, an die Stelle der Totzeit tritt hier jedoch die Abtastzeit. Ist die Option *Abtastzeit in alle zeitdiskreten Blöcke übernehmen* aktiviert, so wird die Abtastzeit des Blocks nach dem Verlassen des Dialogs automatisch in alle anderen zeitdiskreten Blöcke (d. h. Blöcke vom Typ UNITDELAY oder ZÜFKT) übernommen.

Die Abtastzeit sollte ein ganzzahliges Vielfaches der Simulationsschrittweite betragen, andernfalls erfolgt zu Beginn der Simulation eine Warnmeldung.

Parameter-grenzen: $m \le n, \;\; n \le 8, \;\; a_n \neq 0$

Statische Blöcke

Lineare Kennlinie mit Offset

Typname: LINSKAL

Funktion: Dieser Block realisiert eine lineare Kennlinie mit Offset (Variablenskalierer) der Form

$$y(t) = ax(t) + b\,.$$

Dabei ist *x* die Eingangsgröße des Blocks, *y* die Ausgangsgröße, *a* die Geradensteigung und *b* der Offset. Der Block kann daher z. B. für das Umskalieren einer Variablen benutzt werden.

Parameterdialog:

Parametergrenzen: keine

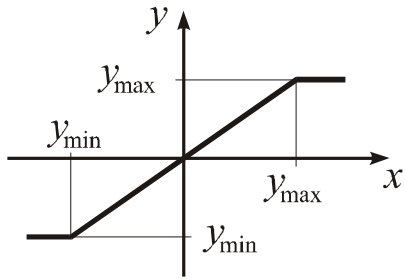
Begrenzer

Typname: BEGRENZER

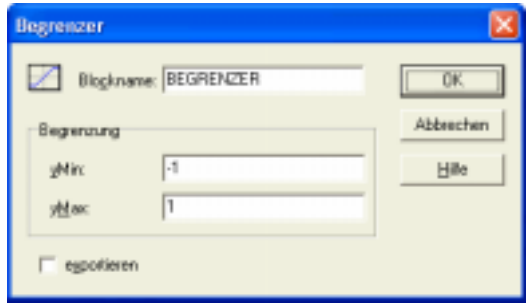
Funktion: Die Begrenzer- oder Sättigungskennlinie begrenzt die Ausgangsgröße auf den Wertebereich $y_{\min} \le y(t) \le y_{\max}$ gemäß der Vorschrift

$$y(t) = \begin{cases} y_{\min} & \text{für } x(t) < y_{\min} \\ x(t) & \text{für } y_{\min} \le x(t) \le y_{\max} \\ y_{\max} & \text{für } x(t) > y_{\max} \end{cases}.$$

Im linearen Arbeitsbereich besitzt das Kennlinienglied die Steigung 1. Die Kennlinie hat demnach die folgende Gestalt:



Parameter-dialog:



Parameter-grenzen: $y_{\min} \le y_{\max}$

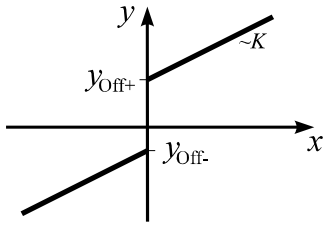
Vorlastkennlinie

Typname: VORLAST

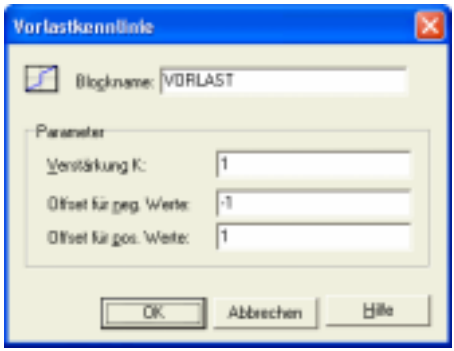
Funktion: Die Vorlastkennlinie ist stückweise linear mit einer Unstetigkeit an der Stelle $x = 0$. Sie wird charakterisiert durch die Beziehung

$$y(t) = \begin{cases} y_{Off-} + K\,x(t) & für \quad x(t) \le 0 \\ y_{Off+} + K\,x(t) & für \quad x(t) > 0 \end{cases}$$

und hat die nachfolgende Gestalt:



Parameter-dialog:



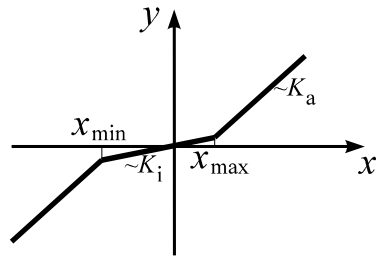
Unempfindlichkeitszone

Typname: UNEMPFINDLICH

Funktion: Dieser Systemtyp realisiert eine Verringerung der Verstärkung von K_a auf K_i innerhalb einer Unempfindlichkeitszone [x_{min}, x_{max}] der Eingangsgröße $x(t)$ gemäß der Beziehung

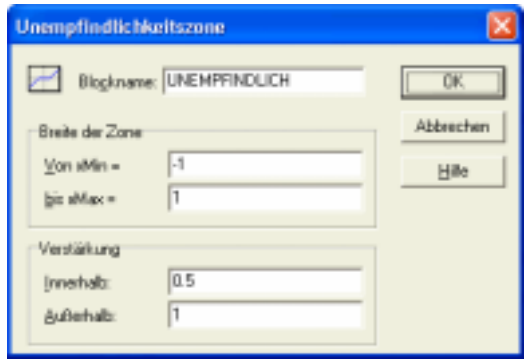
$$y(t) = \begin{cases} K_a\,x(t) + x_{min}\,(K_i - K_a) & für\, x(t) < x_{min} \\ K_i\,x(t) & für\, x_{min} \le x(t) \le x_{max} \\ K_a\,x(t) + x_{max}\,(K_i - K_a) & für\, x(t) > x_{max} \end{cases} .$$

Die Kennlinie hat die folgende Gestalt:



Für $K_i = 0$ erhält man eine Kennlinie mit toter Zone.

Parameter-dialog:



Parameter-grenzen: $x_{min} \leq x_{max}$

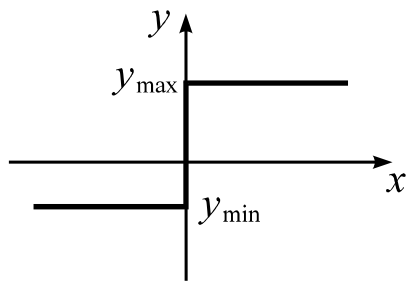
Zweipunktglied

Typname: ZWEIPUNKT

Funktion: Die Kennlinie gehorcht der Beziehung

$$y(t) = \begin{cases} y_{min} & \textit{für} \quad x(t) < 0 \\ y_{max} & \textit{für} \quad x(t) \geq 0 \end{cases}$$

und hat die folgende Gestalt:



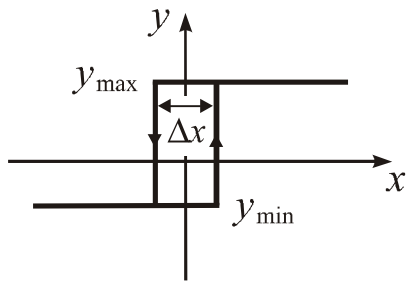
Parameter-dialog:

Parameter-grenzen: $y_{\min} \leq y_{\max}$

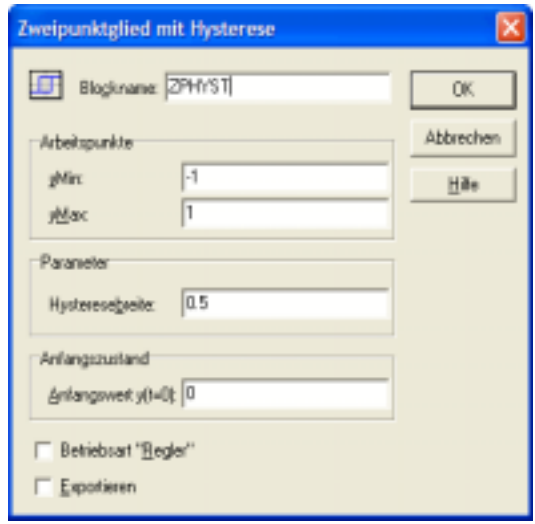
Zweipunktglied mit Hysterese

Typname: ZPHYST

Funktion: Stellt eine Kombination aus einem Zweipunktglied mit den Arbeitspunkten $y_{\min}$ und $y_{\max}$ und einem Hystereseglied mit der Hysteresebreite Δx dar. Nachfolgende Grafik zeigt die zugehörige Kennlinie.



Parameter-dialog:



In der Betriebsart *Regler* weist der Block zwei Eingangsgrößen (Führungsgröße *w* und Rückführgröße *r*) auf; die Regeldifferenz wird in diesem Fall blockintern aus der Differenz dieser beiden Größen berechnet.

Parameter-grenzen: $y_{min} \leq y_{max}$

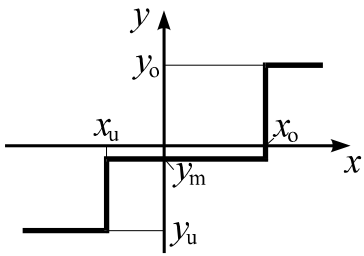
Dreipunktglied

Typname: DREIPUNKT

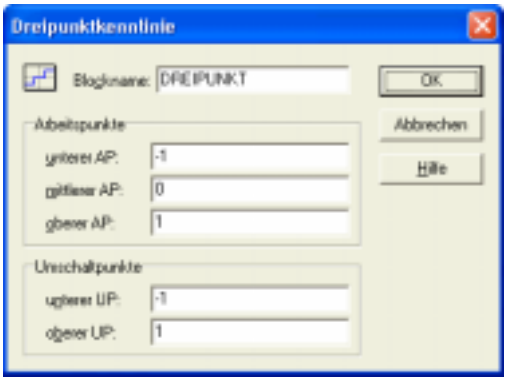
Funktion: Die Kennlinie gehorcht der Beziehung

$$y(t) = \begin{cases} y_u & \text{für } x(t) < x_u \\ y_m & \text{für } x_u \le x(t) \le x_o \\ y_o & \text{für } x(t) > x_o \end{cases}$$

und hat die folgende Gestalt:



Parameter-dialog:

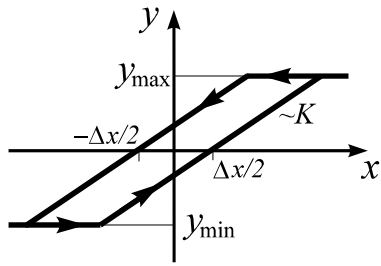


Parameter-grenzen: $x_u \le x_o, y_u \le y_m \le y_o$

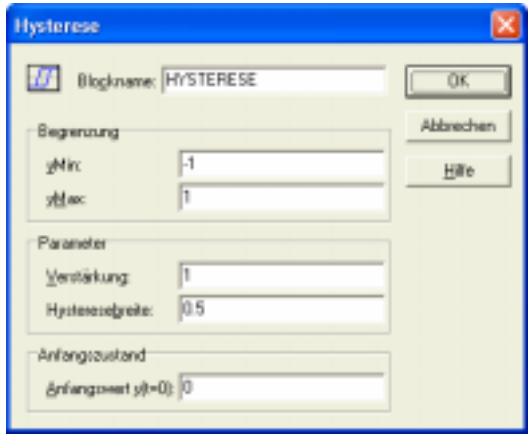
Hysteresekennlinie

Typname: HYSTERESE

Funktion: Die Hysteresekennlinie wird durch Verstärkung K, Hysteresebreite Δx und Begrenzung y_{min} bzw. y_{max} charakterisiert und besitzt folgende Gestalt:



Parameter-dialog:



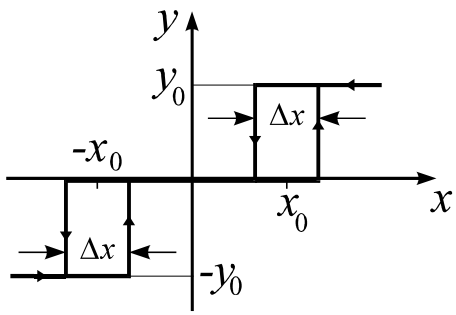
Zur Simulation muss der Anfangswert $y(t=0)$ vorgegeben werden.

Parameter-grenzen: $K, \Delta x > 0,\ y_{\min} \le y_{\max}$

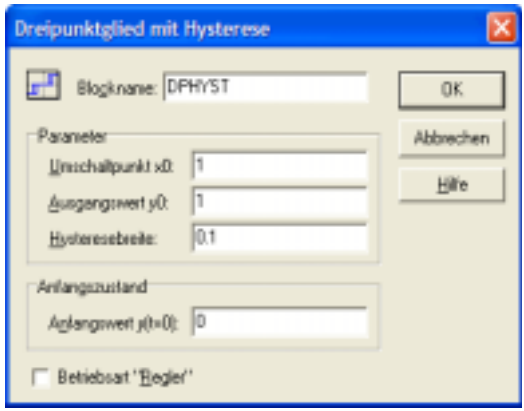
Dreipunktglied mit Hysterese

Typname: DPHYST

Funktion: Stellt eine Kombination aus einem symmetrischen Dreipunktglied mit den Umschaltpunkten x_0 und $-x_0$ und den Arbeitspunkten y_0 und $-y_0$ und einem Hystereseglied mit der Hysteresebreite Δx dar.



Parameter-dialog:



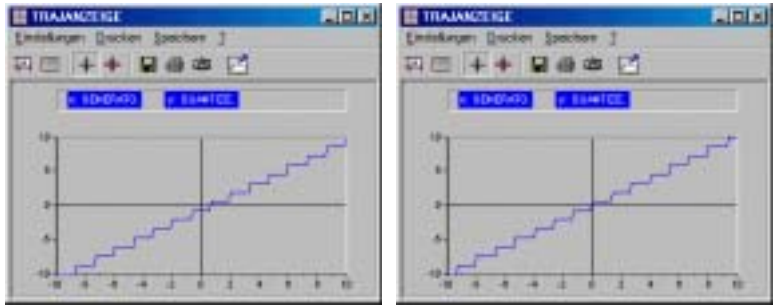
In der Betriebsart *Regler* weist der Block zwei Eingangsgrößen (Führungsgröße *w* und Rückführgröße *r*) auf; die Regeldifferenz wird in diesem Fall blockintern aus der Differenz dieser beiden Größen berechnet.

Parameter-grenzen: $x_0, y_0, \Delta x > 0$

Quantisierer

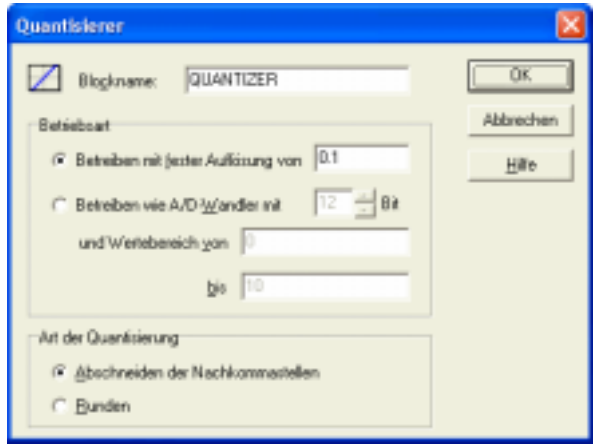
Typname: QUANTIZER

Funktion: Dieser Blocktyp realisiert einen Quantisierer, der entweder mit einer festen Auflösung oder wie ein *n*-Bit-A/D-Wandler betrieben werden kann. Die Quantisierung kann entweder durch Abschneiden der Nachkommastellen oder durch Runden erfolgen.



Beispiel: Kennlinie eines als 4-Bit-A/D-Wandler (Wertebereich -10 ... +10) betriebenen Quantisierers mit Abschneiden der Nachkommastellen (links) bzw. Runden (rechts)

Parameter-dialog:



Benutzerdefinierte Kennlinie

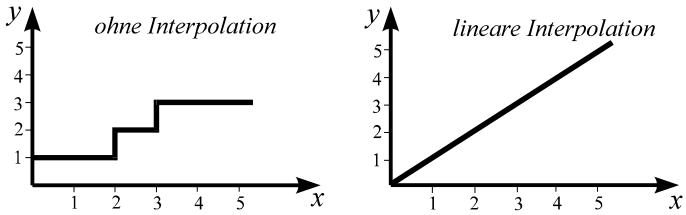
Typname: KENNLINIE

Funktion: Ermöglicht die Definition einer Kennlinie $y(x)$ über n Wertepaare (x_i, y_i). Die Stützstellen x_i können beliebig (d. h. nicht zwangsläufig äquidistant), müssen aber in aufsteigender Reihenfolge sortiert sein. Zwischen den eingelesenen Werten kann linear interpoliert werden. Andernfalls wird der letzte gültige Wert jeweils beibehalten, bis ein neuer Wert auftritt. In diesem Fall ergibt sich eine stufenförmige Kennlinie.

Beispiel: Die Kennlinie sei definiert über die drei Wertepaare

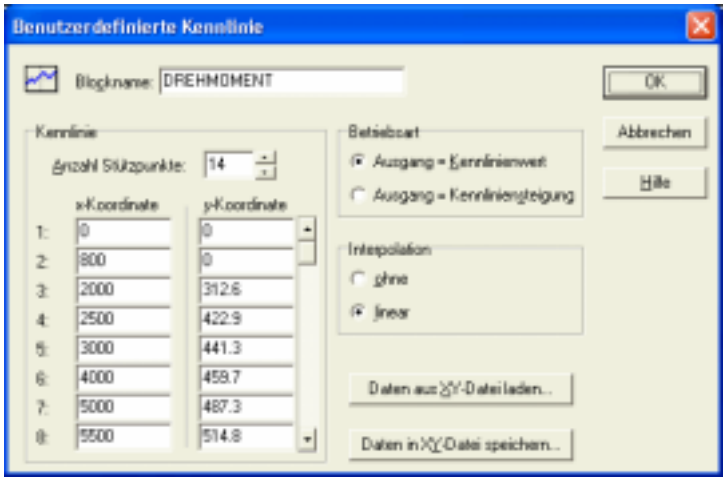
(1, 1), (2, 2), (3, 3).

Es ergeben sich dann folgende Kennlinienverläufe:



Statt des Ordinatenwertes selbst kann auch die *Steigung* dy/dx der Kennlinie für den am Eingang anliegenden Wert ermittelt und ausgegeben werden. In diesem Fall ist eine ggf. zusätzlich gewählte Interpolation ohne Wirkung. Für obige Beispielkennlinie ergäbe sich z. B. für alle Eingangswerte eine konstante Steigung von eins.

Parameter-dialog:



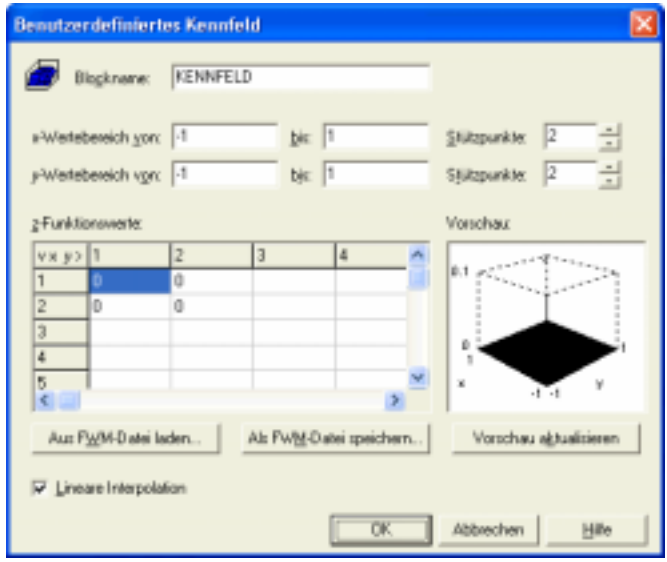
Parameter-grenzen: $2 \le n \le 1000$

Benutzerdefiniertes Kennfeld

Typname: KENNFELD

Funktion: Ermöglicht die Definition eines Kennfelds $z(x, y)$ über eine $n \times m$-Stützstellenmatrix (n, $m \leq 100$). Die Matrix kann wahlweise über Tastatur eingegeben oder aus einer FWM-Datei eingelesen werden (siehe Kapitel *WinFACT-Dateiformate*). Standardmäßig wird zwischen den Stützstellen linear interpoliert; diese Option ist bei Bedarf abschaltbar. Außerhalb des Definitionsbereichs der Stützstellenmatrix wird bei aktivierter Interpolation extrapoliert; liegen die aktuellen Block-Eingangswerte außerhalb des Definitionsbereichs, so wird dies durch einen HIGH-Pegel am Ausgang *Ex* des Blocks angezeigt.

Parameterdialog:



Regler

PID-Regler

siehe Blockgruppe *Dynamische Blöcke*

Adaptiver PID-Regler mit steuerbarer Begrenzung

siehe Blockgruppe *Dynamische Blöcke*

Industrie-PID-Regler

siehe Blockgruppe *Aktionsblöcke*

Zweipunktglied

siehe Blockgruppe *Statische Blöcke*

Zweipunktglied mit Hysterese

siehe Blockgruppe *Statische Blöcke*

Dreipunktglied

siehe Blockgruppe *Statische Blöcke*

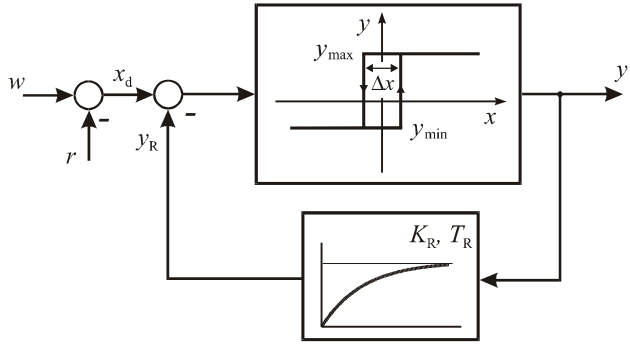
Dreipunktglied mit Hysterese

siehe Blockgruppe *Statische Blöcke*

Zweipunktregler mit PT_1-Rückführung

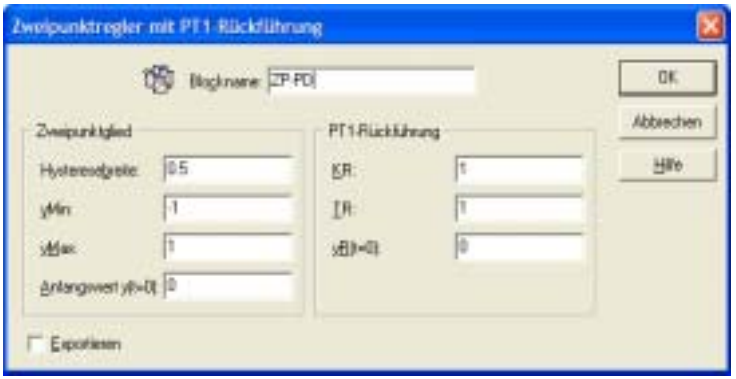
Typname: ZP-PD

Funktion: Dieser Block stellt ein Zweipunktglied mit den Arbeitspunkten y_{min} und y_{max} und der Hysteresebreite Δx dar, bei dem eine Rückführung der Stellgröße über ein PT_1-Glied mit der Verstärkung K_R und der Zeitkonstanten T_R erfolgt. Der Regler erhält dadurch bei entsprechender Parametrierung PD-ähnliches Verhalten. Nachfolgende Grafik zeigt die Struktur des Reglers.



Die Blockausgänge *y+* und *y-* (zweiter bzw. dritter Blockausgang) können als Schaltausgänge benutzt werden; sie weisen jeweils HIGH-Pegel auf, wenn sich das Zweipunktglied im oberen bzw. unteren Arbeitspunkt befindet.

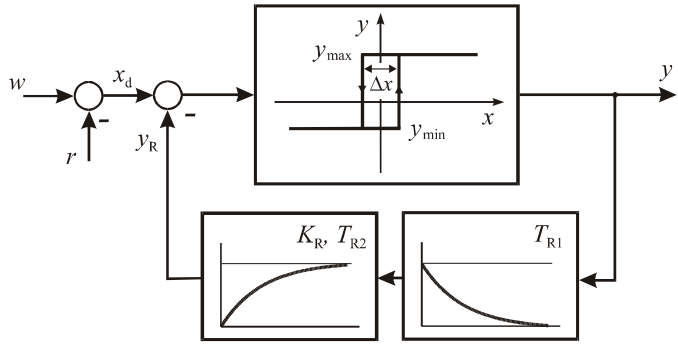
Parameter-dialog:



Zweipunktregler mit verzögert nachgebender Rückführung

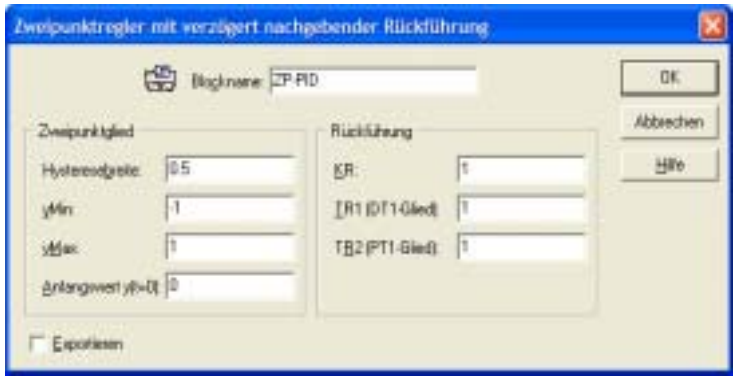
Typname: ZP-PID

Funktion: Dieser Block stellt ein Zweipunktglied mit den Arbeitspunkten y_{min} und y_{max} und der Hysteresebreite Δx dar, bei dem eine Rückführung der Stellgröße über eine Reihenschaltung aus einem DT_1-Glied (Zeitkonstante T_{R1}) und einem PT_1-Glied (Verstärkung K_R, Zeitkonstante T_{R2}) erfolgt. Der Regler erhält dadurch bei entsprechender Parametrierung PID-ähnliches Verhalten. Nachfolgende Grafik zeigt die Struktur des Reglers.



Die Blockausgänge *y+* und *y-* (zweiter bzw. dritter Blockausgang) können als Schaltausgänge benutzt werden; sie weisen jeweils HIGH-Pegel auf, wenn sich das Zweipunktglied im oberen bzw. unteren Arbeitspunkt befindet.

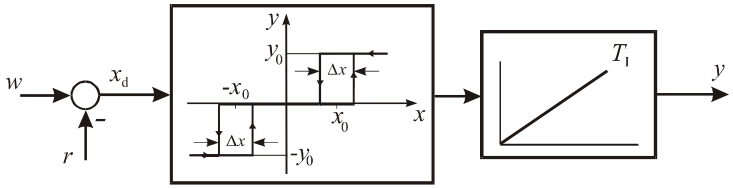
Parameter-dialog:



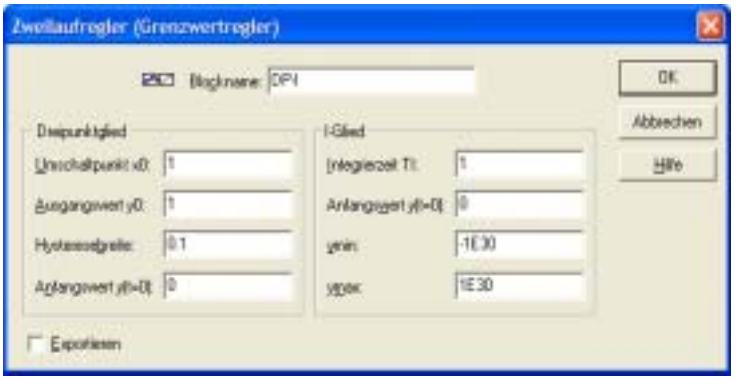
Zweilaufregler (Grenzwertregler)

Typname: DP-I

Funktion: Dieser Block stellt ein symmetrisches Dreipunktglied mit den Umschaltpunkten x_0 und $-x_0$, den Arbeitspunkten y_0 und $-y_0$ und der Hysteresebreite Δx dar, dem ein auf den Bereich $[y_{min}, y_{max}]$ begrenzter Integrierer mit der Integrationszeitkonstanten T_I nachgeschaltet ist. Der Regler ist dadurch in der Lage, einen stetigen Stellgrößenverlauf zu generieren. Nachfolgende Grafik zeigt die Struktur des Reglers.



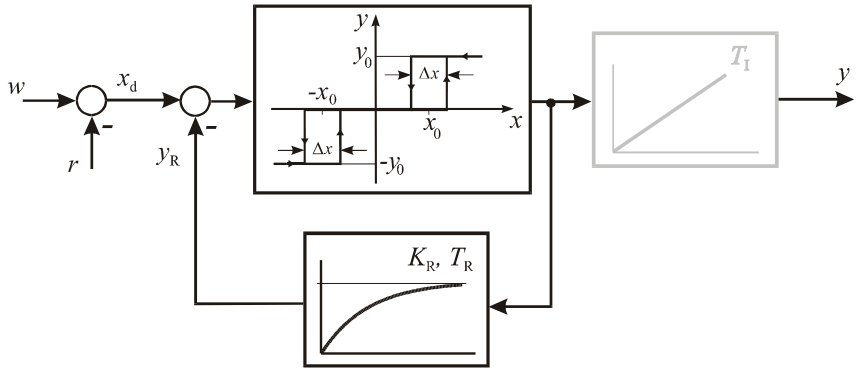
Parameter-dialog:



Dreipunkt-Schrittregler

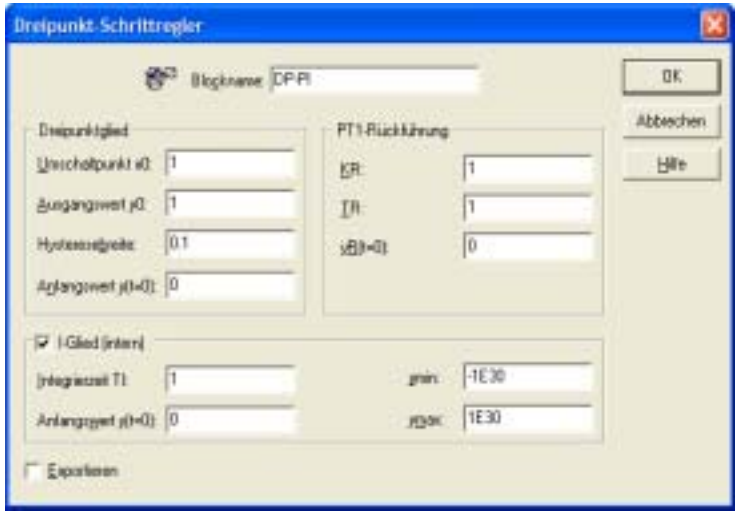
Typname: DP-PI

Funktion: Dieser Block stellt ein symmetrisches Dreipunktglied mit den Umschaltpunkten x_0 und $-x_0$, den Arbeitspunkten y_0 und $-y_0$ und der Hysteresebreite Δx dar, dessen Ausgangsgröße über ein PT_1-Glied mit der Verstärkung K_R und der Zeitkonstanten T_R zurückgekoppelt ist und dem optional ein auf den Bereich $[y_{min}, y_{max}]$ begrenzter Integrierer mit der Integrationszeitkonstanten T_I nachgeschaltet ist. Der Regler weist dadurch bei entsprechender Parametrierung PI-ähnliches Verhalten auf. Nachfolgende Grafik zeigt die Struktur des Reglers.



Die Blockausgänge *y+* und *y-* (zweiter bzw. dritter Blockausgang) können als Schaltausgänge benutzt werden; sie weisen jeweils HIGH-Pegel auf, wenn sich das Dreipunktglied im oberen bzw. unteren Arbeitspunkt befindet.

Parameter-dialog:



Fuzzy Controller

siehe Blockgruppe *Sonstige Systemblöcke*

Online-Fuzzy Controller

siehe Blockgruppe *Sonstige Systemblöcke*

Stellglieder

Stellglied Typ I

Typname: STELLGLIED_1

Funktion: Dieser Block stellt ein Stellglied dar, das neben der typischen Ausgangsgrößenbegrenzung

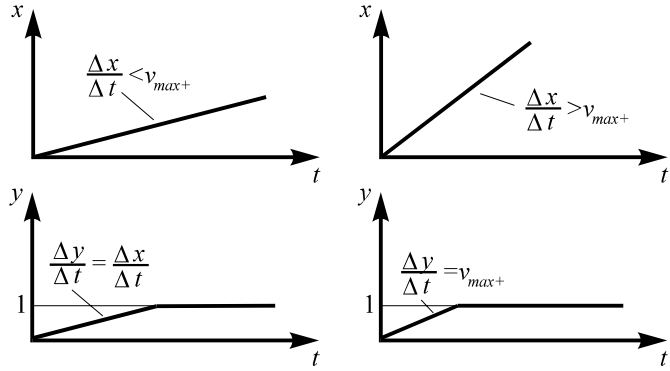
$$y(t) = \begin{cases} y_{\min} & \text{für } x(t) < y_{\min} \\ x(t) & \text{für } y_{\min} \le x(t) \le y_{\max} \\ y_{\max} & \text{für } x(t) > y_{\max} \end{cases}$$

(vgl. Begrenzer-Block) eine Anstiegsgeschwindigkeitsbegrenzung besitzt. Die Änderungsgeschwindigkeit der Ausgangsgröße des Stellglieds wird dabei für negative Änderungen auf $v_{\max-}$ und für positive Änderungen auf $v_{\max+}$ begrenzt:

$$v_{\max-} \le \frac{\mathrm{d}y}{\mathrm{d}t} \le v_{\max+}$$

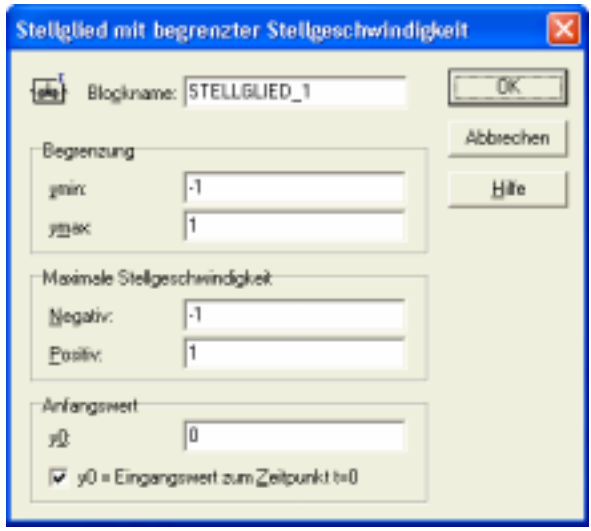
Nachfolgendes Bild zeigt den Verlauf der Ausgangsgröße bei verschiedenen Eingangsrampen für den Fall

$$y_{\min} = -1, y_{\max} = 1, \; v_{\max-} = -1, v_{\max+} = 1 .$$



Verhalten des Stellglieds für eine Eingangsrampe mit kleiner (links) und großer Steigung, d. h. Änderungsgeschwindigkeit (rechts)

Parameter-dialog:



Parameter-grenzen: $y_{\min} < y_{\max}$, $v_{\max-} < 0, v_{\max+} > 0$

Stellglied Typ II

Typname: STELLGLIED_2

Funktion: Dieser Stellglied-Typ weist wie Typ I eine Ausgangsgrößenbegrenzung auf, besitzt jedoch eine *konstante* Anstiegsgeschwindigkeit: Für negative Eingangsgrößenänderungen ändert sich die Ausgangsgröße unabhängig vom Betrag der Eingangsgrößenänderung mit v_-, für positive Eingangsgrößenänderungen mit v_+. Außerdem besitzt das Stellglied eine tote Zone der Breite Δx. Eingangsgrößen, die betragsmäßig kleiner sind als $\Delta x / 2$, werden ignoriert, d. h. der aktuelle Ausgangswert wird beibehalten.

Parameterdialog:

Parametergrenzen:

$$y_{\min} < y_{\max}$$
$$v_- < 0, \quad v_+ > 0$$
$$\Delta x / 2 \geq 0$$

Funktionsblöcke

Verknüpfer

Typname: VERKNUEPFER

Funktion: Dieser Block erlaubt die Verknüpfung von zwei bis 50 Eingangsgrößen über die Operationen Summation, Multiplikation oder Division. Für jede Eingangsgröße x_i kann getrennt eine Vorzeichenumkehr erfolgen. Es gilt:

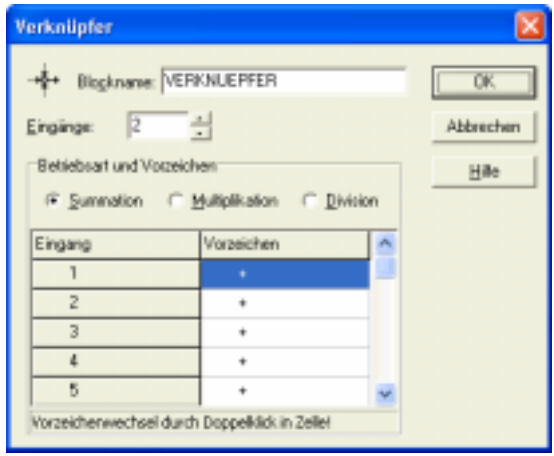
Betriebsart *Summation*: $y = \pm x_1 \pm x_2 \pm x_3 \pm ...$

Betriebsart *Multiplikation*: $y = (\pm x_1) \cdot (\pm x_2) \cdot (\pm x_3) \cdot ...$

Betriebsart *Division*: $y = (\pm x_1) / (\pm x_2) / (\pm x_3) / ...$

In den Betriebsarten *Multiplikation* und *Division* werden etwaige offene Eingänge zu eins gesetzt. In der Betriebsart *Summation* werden die Block-Eingangsfelder mit dem jeweiligen Vorzeichen der Eingangsgröße gekennzeichnet.

Parameterdialog:



Funktion einer Veränderlichen

Typname: FKT1

Funktion: Dieser Block erlaubt die Definition einer Funktion $y = f(x)$. Zur Auswahl stehen folgende Funktionen:

$y = \sin(x)$ $y = \cos(x)$ $y = \tan(x)$ $y = \exp(x)$ $y = \ln(x)$

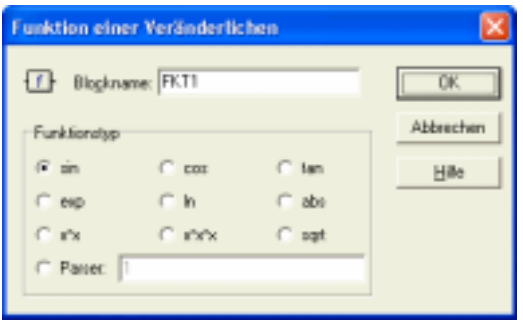
$y = |x|$ $y = x^2$ $y = x^3$ $y = \sqrt{x}$

Alternativ dazu kann die Funktion vom Anwender frei definiert und über einen Funktionsparser interpretiert werden (siehe dazu Blockbeschreibung *Generator*). Als unabhängige Variable ist in diesem Fall "x" anzugeben.

Beispiel: *atan(x)+asin(x)+5*

Die Interpretation der Funktion über den Parser kann je nach Komplexität der Funktion sehr rechenzeitaufwendig sein.

Parameterdialog:



Funktion zweier Veränderlicher

Typname: FKT2

Funktion: Dieser Block erlaubt die Definition einer Funktion $z = f(x, y)$, wobei x und y Eingangsgrößen des Blocks und z seine Ausgangsgröße ist. Zur Auswahl stehen folgende Funktionen:

$z = x + y$ $z = x - y$ $z = xy$ $z = x / y$ $z = x^y$ $z = x^{1/y}$

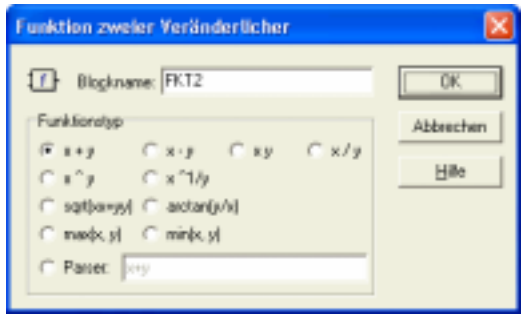
$z = \sqrt{x^2 + y^2}$ $z = \arctan(y / x)$ $z = \max(x, y)$ $z = \min(x, y)$

Alternativ dazu kann die Funktion vom Anwender frei definiert und über einen Funktionsparser interpretiert werden (siehe dazu Blockbeschreibung *Generator*). Als unabhängige Variablen sind in diesem Fall "x" und "y" anzugeben.

Beispiel: *sin(x) + cos(y)*

Die Interpretation der Funktion über den Parser kann je nach Komplexität der Funktion sehr rechenzeitaufwendig sein.

Parameterdialog:



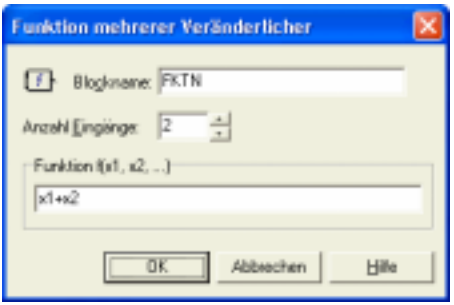
Funktion mehrerer Veränderlicher

Typname: FKTN

Funktion: Dieser Block erlaubt die Definition einer Funktion mit bis zu 50 Veränderlichen über einen Funktionsparser (siehe dazu Blockbeschreibung *Generator*). Als unabhängige Variablen sind in diesem Fall "x1" bis "x10" anzugeben.

Beispiel: $x1 + x2 + x3 - x4 * x5$

Parameter-dialog:



Extremwertbestimmung

Typname: EXTREMWERT

Funktion: Dieser Block erlaubt die Bestimmung des Extremwerts der Eingangsgröße $x(t)$. Es sind vier verschiedene Betriebsarten möglich:

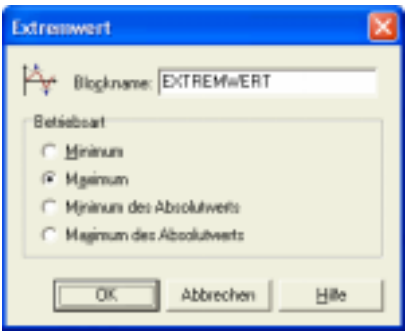
Betriebsart *Minimum*: $y(t) = \min_{0<t<T_{Simu}} x(t)$

Betriebsart *Maximum*: $y(t) = \max_{0<t<T_{Simu}} x(t)$

Betriebsart *Betragsminimum*: $y(t) = \min_{0<t<T_{Simu}} |x(t)|$

Betriebsart *Betragsmaximum*: $y(t) = \max_{0<t<T_{Simu}} |x(t)|$

Parameter-dialog:

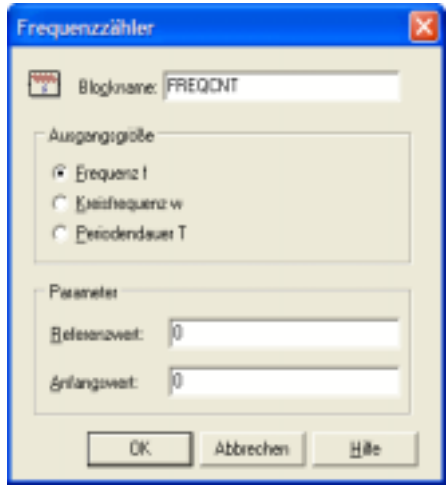


Frequenzzähler

Typname: FREQCNT

Funktion: Dieser Block erlaubt die Ermittlung von Frequenz, Kreisfrequenz oder Periodendauer einer periodischen Funktion anhand aufeinander folgender Nulldurchgänge des auf einen Referenzwert bezogenen Eingangssignals. Solange noch keine volle Periode des Eingangssignals ermittelt werden konnte, wird der Blockausgang auf den im Parameterdialog unter *Anfangswert* spezifizierten Wert gesetzt.

Parameterdialog:

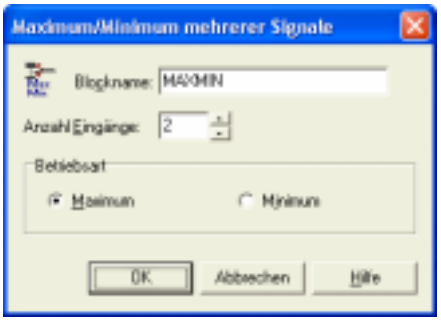


Minimum-/Maximumbestimmung

Typname: MAXMIN

Funktion: Dieser Block schaltet entweder das Minimum oder das Maximum aller Eingangssignale auf den Ausgang durch.

Parameter-dialog:



Statistikfunktionen

Typname: STATISTIK

Funktion: Dieser Block erlaubt die Berechnung statistischer Kennwerte der Eingangsgröße $x(t)$. Als Statistikwerte bezogen auf alle bis zum *k*-ten Simulationsschritt anliegenden Werte sind verfügbar:

Mittelwert: $$y(t_k) = \frac{1}{k}\sum_{i=1}^{k} x(t_i)$$

Mittelwert des Betrags: $$y(t_k) = \frac{1}{k}\sum_{i=1}^{k} \left|x(t_i)\right|$$

Effektivwert: $$y(t_k) = \sqrt{\frac{1}{k}\sum_{i=1}^{k} x^2(t_i)}$$

Standardabweichung: $$y(t_k) = \sqrt{\frac{1}{k}\left(\sum_{i=1}^{k} x^2(t_i) - \frac{1}{k}\left(\sum_{i=1}^{k} x(t_i)\right)^2\right)}$$

Weiterhin sind verschiedene *gleitende* Funktionen verfügbar, die sich nur auf ein bestimmtes *Zeitfenster* (Zeitspanne) beziehen, das die letzten *n* Werte umfasst. Wird beispielsweise eine Zeitspanne von 5 bei einer Simulationsschrittweite von 0.1 vorgegeben, so bestimmen sich die gleitenden Kennwerte jeweils aus den letzten 5/0.1 = 50 Eingangswerten. Folgende Kennwerte sind verfügbar:

Gleitender Mittelwert: $$y(t_k) = \frac{1}{n} \sum_{i=k-n+1}^{k} x(t_i)$$

Gleitender Effektivwert: $$y(t_k) = \sqrt{\frac{1}{n} \sum_{i=k-n+1}^{k} x^2(t_i)}$$

Gleitende Standardabweichung:

$$y(t_k) = \sqrt{\frac{1}{n}\left(\sum_{i=k-n+1}^{k} x^2(t_i) - \frac{1}{n}\left(\sum_{i=k-n+1}^{k} x(t_i) \right)^2 \right)}$$

Gleitende Summe: $$y(t_k) = \sum_{i=k-n+1}^{k} x(t_i)$$

Gleitende Summe der Quadrate: $$y(t_k) = \sum_{i=k-n+1}^{k} x^2(t_i)$$

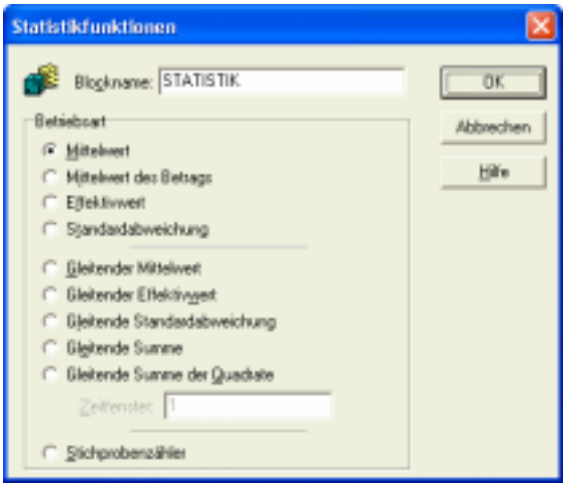
Schließlich kann der Block auch als *Stichprobenzähler* betrieben werden:

Stichprobenzähler: $y(t_k) = k$

Hinweis: Die Gleichungen für die gleitenden Kennwerte gelten erst im "eingeschwungenen" Zustand: Zu Beginn der Simulation - solange noch keine *n* Werte zur Verfügung stehen, also während der ersten *n*-1 Schritte - werden alle bis dahin vorliegenden Eingangswerte zur Ermittlung der gleitenden Kenngrößen benutzt! Man beachte weiterhin, dass in allen Betriebsarten der Eingangswert zum Zeitpunkt $t = 0$ mitgerechnet wird! Nach dem ersten "echten" Simulationsschritt ist also $k = 2$!

Der Block kann über eine positive Flanke am Rücksetzeingang R jederzeit zurückgesetzt werden. Die Berechnung der statistischen Kennwerte erfolgt dann ab diesem Zeitpunkt neu.

Parameter-dialog:



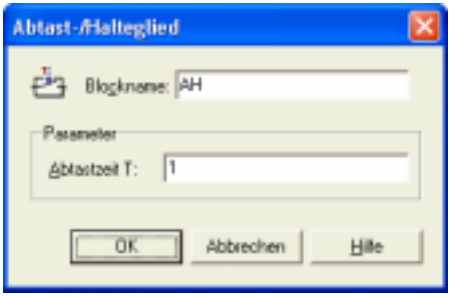
Parameter-grenzen: Keine Einschränkungen

Abtast-/Halteglied

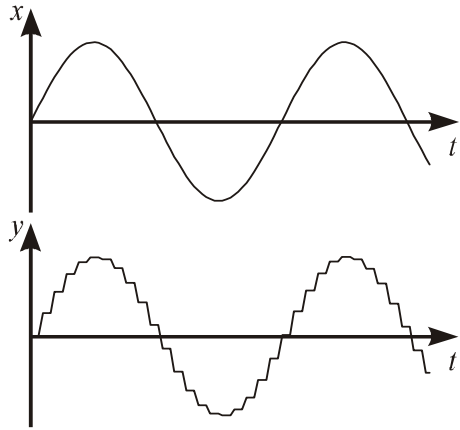
Typname: AH

Funktion: Dieser Funktionsblock realisiert ein Abtast-Halteglied nullter Ordnung: Die Eingangsgröße wird in einem vorgebbaren Zeitabstand T abgetastet, auf den Ausgang geschaltet und dort bis zum nächsten Abtastzeitpunkt konstantgehalten. Die Abtastzeit T sollte ein ganzzahliges Vielfaches der Simulationsschrittweite ΔT betragen.

Parameter-dialog:



Beispiel: Nachfolgende Grafik zeigt den Ausgangsgrößenverlauf eines Abtast-/Haltegliedes bei sinusförmiger Eingangsgröße.



Parametergrenzen: $T > 0$

Triggerbares Abtast-/Halteglied

Typname: EXTERNAH

Funktion: Dieser Funktionsblock realisiert ein Abtast-Halteglied, bei dem der Abtastzeitpunkt durch eine positive Flanke am Takteingang C vorgegeben wird.

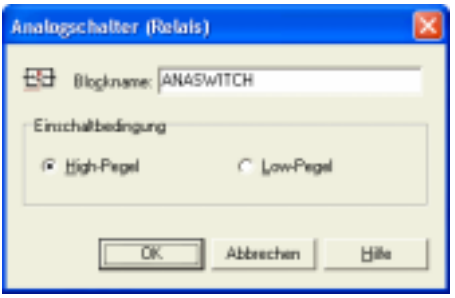
Parameterdialog:

Analogschalter (Relais)

Typname: ANASWITCH

Funktion: Dieser Funktionsblock bildet einen Analogschalter, d. h. ein Relais, nach: Die Eingangsgröße $x(t)$ wird auf den Ausgang geschaltet, wenn am Steuereingang S High-Pegel bzw. Low-Pegel (umschaltbar) anliegt. Andernfalls wird die Ausgangsgröße auf null gesetzt.

Parameter-dialog:

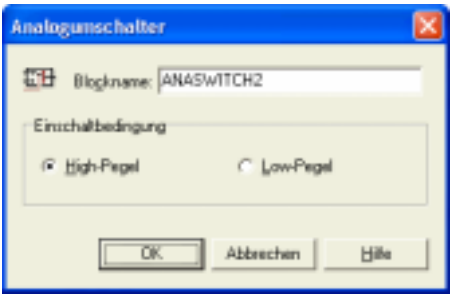


Analogumschalter

Typname: ANASWITCH2

Funktion: Dieser Funktionsblock realisiert einen Analogumschalter. Die Eingangsgröße $x_1(t)$ wird auf den Ausgang geschaltet, wenn am Steuereingang S High-Pegel bzw. Low-Pegel (umschaltbar) anliegt. Andernfalls wird die Ausgangsgröße auf $x_2(t)$ gesetzt.

Parameter-dialog:

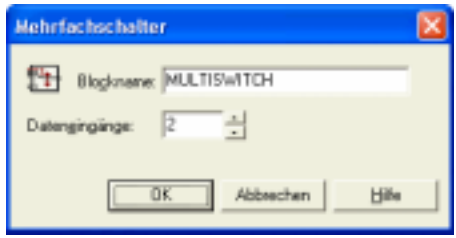


Mehrfachschalter

Typname: MULTISWITCH

Funktion: Dieser Funktionsblock realisiert einen Mehrfachschalter, der über den Steuereingang S geschaltet wird. Der an S anliegende (ggfs. gerundete) Signalwert bestimmt den Dateneingang, der auf den Ausgang durchgeschaltet wird. Beispiel: Liegt an S ein Wert von 2 an, wird der zweite Dateneingang auf den Ausgang gelegt. Liegt S nicht im gültigen Bereich, wird der erste Dateneingang durchgeschaltet.

Parameterdialog:



Digitalbausteine

Logikgatter mit einem Eingang

Typname: LOGIK1

Funktion: Dieser Block stellt ein Logikgatter mit einem Eingang dar. Es sind drei unterschiedliche Betriebsarten möglich:

Negation:	$y = \bar{x}$	(Eingangssignal wird negiert)
TRUE:	$y = 1$	(Ausgang hat immer High-Pegel)
FALSE:	$y = 0$	(Ausgang hat immer Low-Pegel)

Parameter-dialog:

Logikgatter mit zwei Eingängen

Typname: LOGIK2

Funktion: Logikgatter mit zwei Eingängen *x* und *y* und dem Ausgang *z*. Es sind folgende Verknüpfungen möglich:

AND: $z = x \wedge y$

OR: $z = x \vee y$

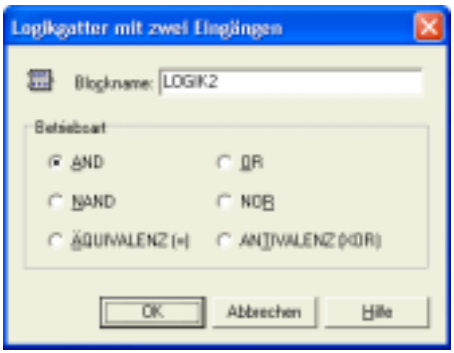
NAND: $z = \overline{x \wedge y}$

NOR: $z = \overline{x \vee y}$

Äquivalenz: $z = x \equiv y$

Antivalenz: $z = x \neq y$

Parameter-dialog:

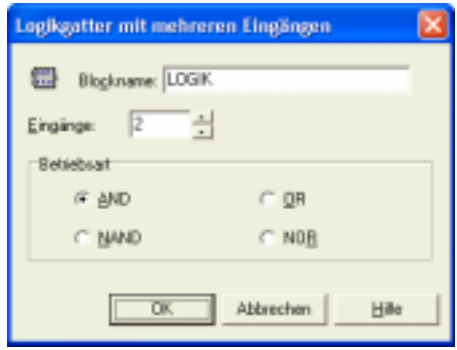


Logikgatter mit mehreren Eingängen

Typname: LOGIK

Funktion: Logikgatter mit bis zu 50 Eingängen und den Betriebsarten *AND*, *NAND*, *OR* und *NOR*.

Parameter-dialog:



RS-Flip-Flop

Typname: RSFLIPFLOP

Funktion: Dieser Block realisiert ein RS-Flip-Flop. Der Ausgang wird durch den Eingang S gesetzt und durch den Eingang R zurückgesetzt. Das Flip-Flop kann in zwei verschiedenen Betriebsarten betrieben werden:

- In der Betriebsart *statisch* sind die statischen Eingangspegel entscheidend.
- In der Betriebsart *dynamisch* ist das Flip-Flop positiv flankengetriggert, d. h. eine Zustandsänderung findet nur bei Eingangsflanken 0 (Low) $\rightarrow$ 1 (High) statt.

Nachfolgend ist die Wertetabelle für das RS-Flip-Flop dargestellt.

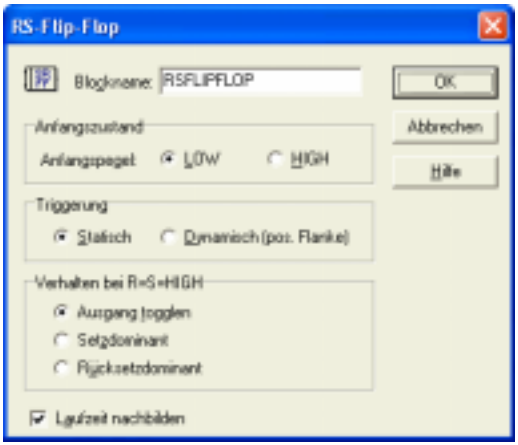
R	S	y_{neu}
0	0	y_{alt}

0	1	1
1	0	0
1	1	$\bar{y}_{alt}$

Der in der Praxis unbestimmte Zustand R = S = 1 (letzte Zeile der Tabelle) führt zu einem Wechsel des Ausgangszustands bei jedem Simulationsschritt.

Für die Simulation muss der Ausgangspegel des Flip-Flops zu Beginn der Simulation vorgegeben werden.

Parameter-dialog:



Ist die Option *Laufzeit nachbilden* aktiviert, so wird der ermittelte Ausgangswert erst einen Simulationsschritt später ausgegeben. Diese Option sollte daher aktiviert werden, wenn mehrere Flip-Flops in Reihe geschaltet werden, z. B. um Schiebe- oder Ringregister nachzubilden oder um z. B. Frequenzteiler aufzubauen (siehe dazu die Beispieldateien SCH_REG.BSY, RING_REG.BSY und TEILER2.BSY im WinFACT-Beispielverzeichnis).

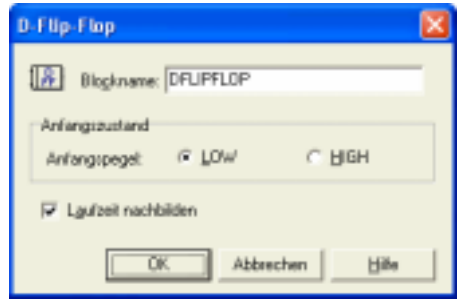
D-Flip-Flop

Typname: DFLIPFLOP

Funktion: Dieser Block stellt ein D-Flip-Flop dar und kann damit als 1-Bit-Speicher oder zur Realisierung von Schieberegistern oder Frequenzteilern verwendet werden. Bei einer positiven Flanke am Takteingang C wird der aktuelle Zustand des Eingangs gespeichert und am Ausgang ausgegeben.

Für die Simulation muss der Ausgangspegel des Flip-Flops zu Beginn der Simulation vorgegeben werden.

Parameterdialog:



Die Einstellung im Feld *Triggerung* ist für diesen Flip-Flop-Typ ohne Bedeutung. Ist die Option *Laufzeit nachbilden* aktiviert, so wird der ermittelte Ausgangswert erst einen Simulationsschritt später ausgegeben. Diese Option sollte daher aktiviert werden, wenn mehrere Flip-Flops in Reihe geschaltet werden, z. B. um Schiebe- oder Ringregister nachzubilden oder um z. B. Frequenzteiler aufzubauen (siehe dazu die Beispieldateien SCH_REG.BSY, RING_REG.BSY und TEILER2.BSY im WinFACT-Beispielverzeichnis).

JK-Flip-Flop

Typname: JKFLIPFLOP

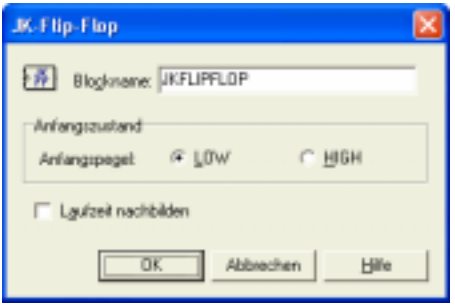
Funktion: Dieser Block realisiert ein positiv-flankengetriggertes JK-Flip-Flop. Bei einer positiven Flanke am Takteingang C wird der Ausgang je nach Zustand des Setzeingangs J und des Rücksetzeingangs K gesetzt bzw. rückgesetzt. Nachfolgend ist die zugehörige Wertetabelle dargestellt.

J	K	y_{neu}
0	0	y_{alt}
0	1	0
1	0	1
1	1	$\bar{y}_{alt}$

Der in der Praxis unbestimmte Zustand J = K = 1 (letzte Zeile der Tabelle) führt zu einem Wechsel des Ausgangszustands bei jedem Simulationsschritt.

Für die Simulation muss der Ausgangspegel des Flip-Flops zu Beginn der Simulation vorgegeben werden.

Parameter-dialog:



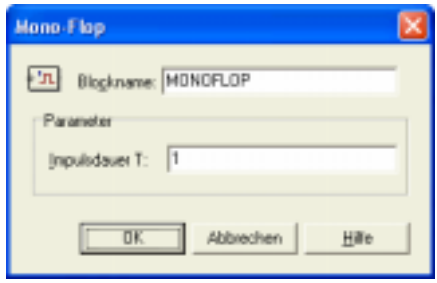
Die Einstellung im Feld *Triggerung* ist für diesen Flip-Flop-Typ ohne Bedeutung. Ist die Option *Laufzeit nachbilden* aktiviert, so wird der ermittelte Ausgangswert erst einen Simulationsschritt später ausgegeben. Diese Option sollte daher aktiviert werden, wenn mehrere Flip-Flops in Reihe geschaltet werden, z. B. um Schiebe- oder Ringregister nachzubilden oder um z. B. Frequenzteiler aufzubauen (siehe dazu die Beispieldateien SCH_REG.BSY, RING_REG.BSY und TEILER2.BSY im WinFACT-Beispielverzeichnis).

Mono-Flop

Typname: MONOFLOP

Funktion: Der Block stellt eine monostabile Kippschaltung (Univibrator) dar. Er erzeugt bei einer positiven Flanke am Eingang am Ausgang einen Impuls vorgebbarer Dauer *T*. Das Mono-Flop ist nicht nachtriggerbar. Trifft ein Impuls am Eingang ein, während der Ausgang noch auf High-Pegel liegt, so wird dieser Impuls ignoriert. Die Impulsdauer sollte ein ganzzahliges Vielfaches der Simulationsschrittweite ΔT betragen.

Parameter-dialog:



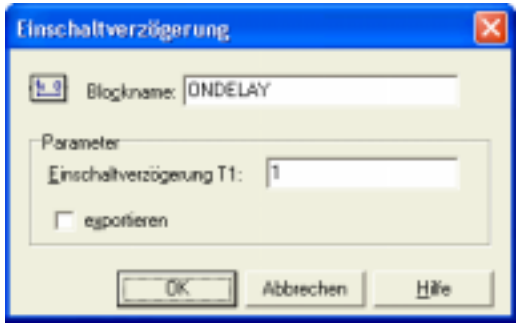
Parameter-grenzen: $T > 0$

Einschaltverzögerung

Typname: ONDELAY

Funktion: Dieser Block verzögert einen Einschaltvorgang (positive Flanke am Blockeingang) um die Zeit T_1.

Parameter-dialog:



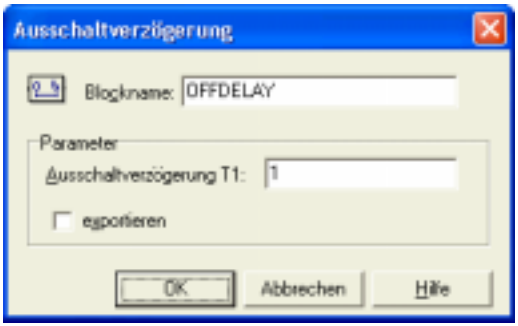
Parameter-grenzen: $T_1 \geq 0$

Ausschaltverzögerung

Typname: OFFDELAY

Funktion: Dieser Block verzögert einen Ausschaltvorgang (negative Flanke am Blockeingang) um die Zeit T_1.

Parameterdialog:



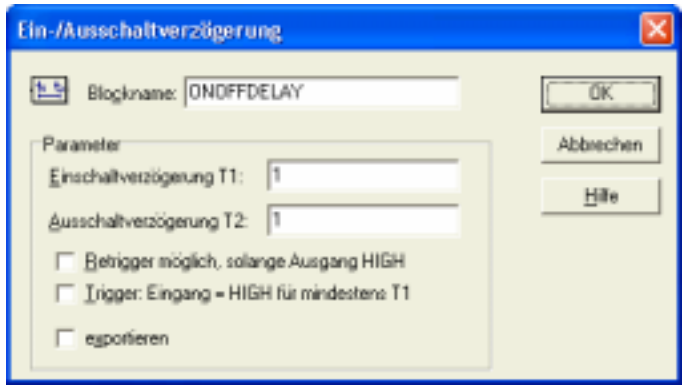
Parametergrenzen: $T_1 \geq 0$

Ein-/Ausschaltverzögerung

Typname: ONOFFDELAY

Funktion: Dieser Block verzögert einen Einschaltvorgang (positive Flanke am Blockeingang) um die Zeit T_1 und einen Ausschaltvorgang (negative Flanke am Blockeingang) um die Zeit T_2.

Parameter-dialog:



Ist die Option *Retrigger möglich, solange Ausgang HIGH* aktiviert, bewirkt eine positive Eingangsflanke im HIGH-Zustand des Ausgangs eine erneute Triggerung des Blocks; andernfalls wird sie ignoriert. Ist die Option *Trigger: Eingang = HIGH für mindestens T1* aktiviert, so wird eine positive Eingangsflanke nur dann nach der Zeit T_1 an den Ausgang übernommen, wenn der Eingang zu dieser Zeit noch HIGH-Pegel aufweist.

Parameter-grenzen: $T_1, T_2 \geq 0$

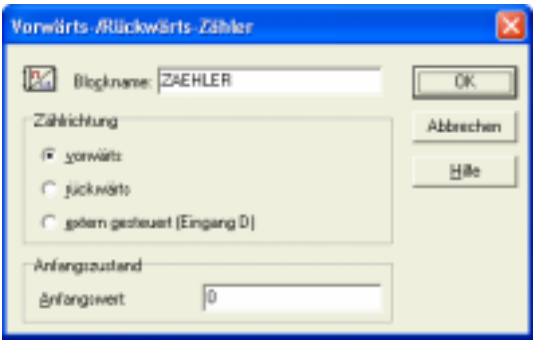
Vorwärts-/Rückwärts-Zähler

Typname: ZAEHLER

Funktion: Dieser Systemblock stellt einen positiv-flankengetriggerten Zählerbaustein mit einem vorgebbaren Anfangswert dar. Die Zählrichtung ist intern oder extern umschaltbar. Der Block kann während der Simulation jederzeit durch eine positive Flanke am Reset-Eingang R auf seinen Anfangswert zurückgesetzt werden.

Der Zähler kann auch negative Zählerstände aufweisen.

Parameter-dialog:



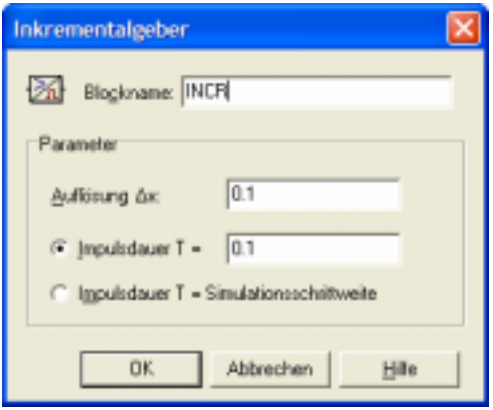
Ist die Option *extern gesteuert* aktiv, so zählt der Baustein bei High-Pegel an Eingang D vorwärts, bei Low-Pegel rückwärts.

Inkrementalgeber

Typname: INCR

Funktion: Dieser Systemblock realisiert einen Inkrementalgeber, d. h. einen Geber zur Erfassung von Lage- oder Winkeländerungen. Der Block erzeugt jeweils bei einer Eingangsgrößenänderung um den Betrag Δx am Ausgang einen Impuls einstellbarer Dauer T.

Parameter-dialog:

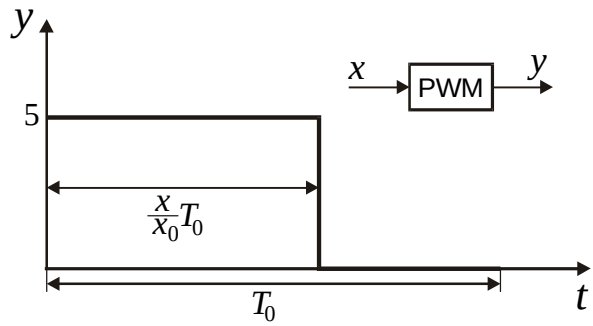


Pulsbreitenmodulator (PWM)

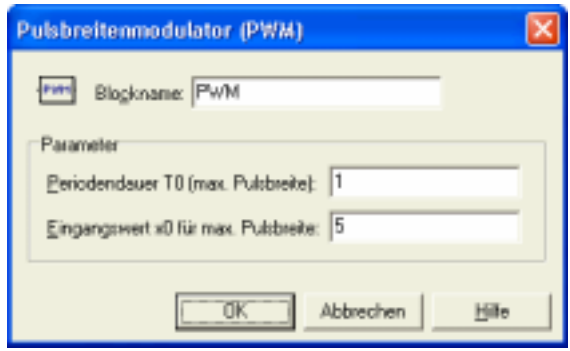
Typname: PWM

Funktion: Dieser Systemblock erzeugt aus einem analogen Eingangssignal *x* ein pulsbreitenmoduliertes Ausgangssignal *y*. Dazu wird am Blockausgang ein HIGH-Puls mit konstanter Taktfrequenz f_0, aber variabler Pulsbreite *T* ausgegeben. Die Pulsbreite ist der Größe des Eingangssignals proportional. Ist $T_0 = 1/f_0$ die Periodendauer des Ausgangssignals und x_0 die maximale zulässige Eingangsgröße des Blocks, so berechnet sich die aktuelle Pulsbreite am Ausgang zu

$$T = \frac{x}{x_0} T_0 .$$



Parameterdialog:



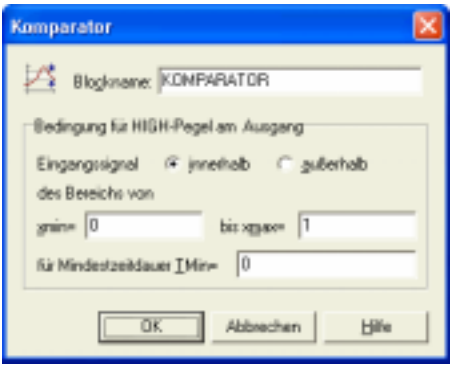
Für eine ordnungsgemäße Funktion sollte die Simulationsschrittweite um ein Vielfaches (mindestens Faktor 10) kleiner als die Periodendauer T_0 sein.

Komparator

Typname: KOMPARATOR

Funktion: Dieser Block realisiert einen Fensterkomparator (Fensterdiskriminator). Er liefert am Ausgang High-Pegel, wenn die Eingangsgröße x für eine vorgebbare Mindestzeitdauer T_{min} innerhalb/außerhalb eines bestimmten Wertebereichs $[\,x_{min}, x_{max}\,]$ liegt. Dadurch ist es möglich, nur kurzzeitig auftretende Bereichsverletzungen zu ignorieren.

Parameterdialog:



Parametergrenzen: $x_{min} < x_{max}, \quad T_{min} \geq 0$

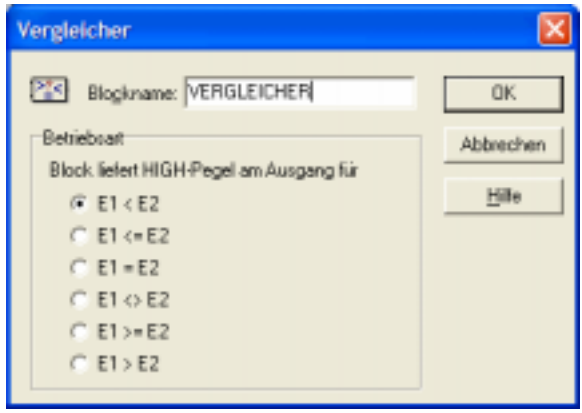
Vergleicher

Typname: VERGLEICHER

Funktion: Der Block vergleicht die beiden analogen Eingangsgrößen E_1 und E_2 und liefert am Ausgang High-Pegel, falls die Vergleichsbedingung erfüllt ist. Es stehen folgende Vergleichsoperatoren zur Auswahl:

$$E_1 < E_2 \quad E_1 \leq E_2 \quad E_1 = E_2 \quad E_1 \geq E_2 \quad E_1 > E_2 \quad E_1 \neq E_2$$

Parameter-dialog:

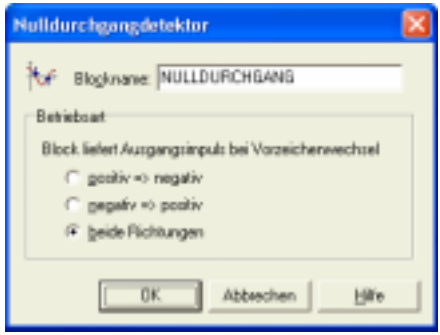


Nulldurchgangdetektor

Typname: NULLDURCHGANG

Funktion: Dieser Block erzeugt einen Ausgangsimpuls, wenn beim Eingangssignal ein Vorzeichenwechsel auftritt. Die Länge des Ausgangsimpulses entspricht der Simulationsschrittweite ΔT. Die Richtung des zu detektierenden Nulldurchgangs kann vorgegeben werden.

Parameter-dialog:

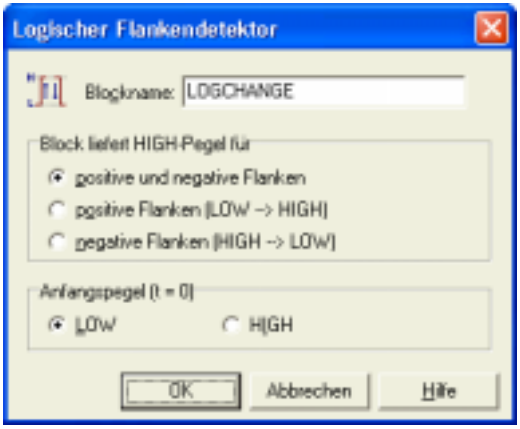


Logischer Flankendetektor

Typname: LOGCHANGE

Funktion: Dieser Block erzeugt einen Ausgangsimpuls, wenn am Eingang ein logischer Pegelwechsel auftritt. Es kann gewählt werden, ob positive Flanken (Wechsel von LOW- auf HIGH-Pegel) und/oder negative Flanken (Wechsel von HIGH- auf LOW-Pegel) detektiert werden sollen. Der Ausgangspegel des Blocks zu Beginn der Simulation ist einstellbar.

Parameter-dialog:

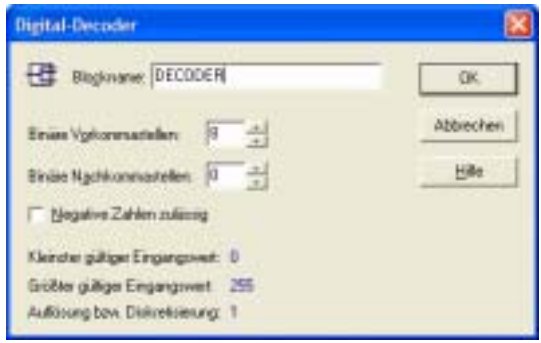


Digital-Decoder

Typname: DECODER

Funktion: Dieser Block wandelt einen am Eingang anliegenden Analogwert in einen Digitalwert um, der in binärer Form an den Blockausgängen erscheint. Die Anzahl der binären Vor- und Nachkommastellen und damit die Diskretisierung ist einstellbar; ebenso können bei Bedarf auch negative Eingangswerte zugelassen werden.

Parameter-dialog:

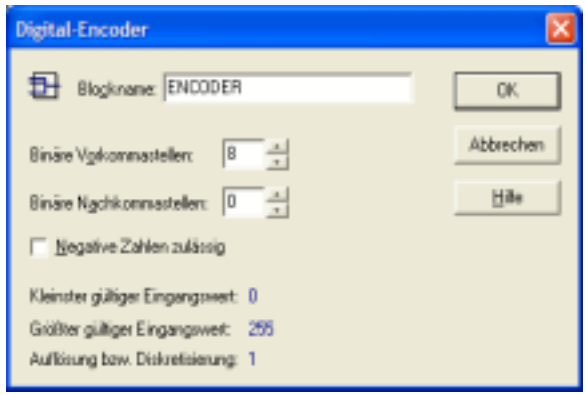


Digital-Encoder

Typname: ENCODER

Funktion: Dieser Block wandelt einen am Eingang in binärer Form anliegenden Digitalwert in einen Analogwert um, der am Blockausgang erscheint. Die Anzahl der binären Vor- und Nachkommastellen und damit die Diskretisierung ist einstellbar; ebenso können bei Bedarf auch negative Eingangswerte zugelassen werden. Der Block stellt das Gegenstück zum Digital-Decoder dar.

Parameter-dialog:

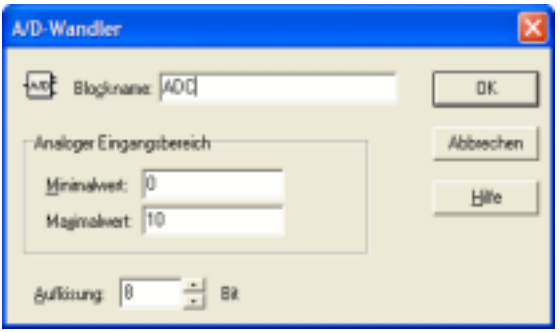


A/D-Wandler

Typname: ADC

Funktion: Dieser Block wandelt einen am Eingang anliegenden Analogwert in einen Digitalwert um, der in binärer Form an den Blockausgängen erscheint. Die Auflösung des Wandlers ist einstellbar. Der Block ähnelt in seiner Funktion dem Block vom Typ *DECODER*.

Parameter-dialog:

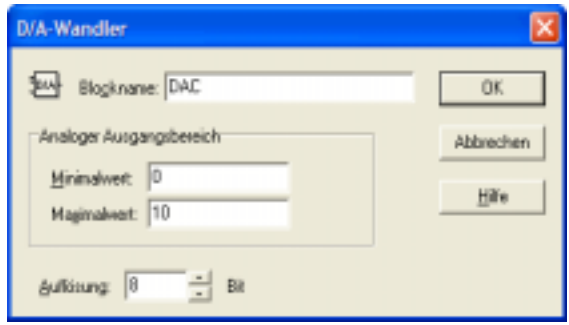


D/A-Wandler

Typname: DAC

Funktion: Dieser Block wandelt einen am Eingang in binärer Form anliegenden Digitalwert in einen Analogwert um, der am Blockausgang erscheint. Die Auflösung des Wandlers ist einstellbar. Der Block ähnelt in seiner Funktion dem Block vom Typ *ENCODER*.

Parameter-dialog:



Aktionsblöcke

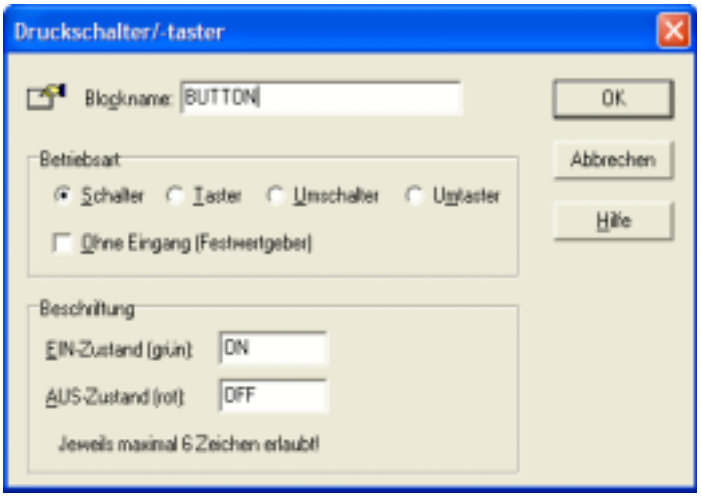
Druckschalter/-taster

Typname: BUTTON

Funktion: Dieser Block realisiert einen vom Anwender mit der linken Maustaste zu betätigenden Druckschalter bzw. –taster mit Ein-/Ausfunktion und wählbarer Beschriftung.

Steuerfenster des Schalters im EIN- bzw. AUS-Zustand

Parameter-dialog:



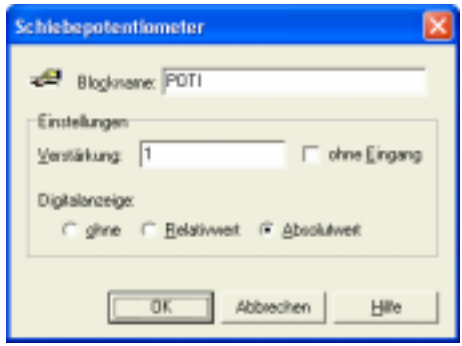
Schieberegler (Potentiometer)

Typname: POTI

Funktion: Dieser Block realisiert einen vom Anwender mit der linken Maustaste zu betätigenden Schieberegler. Der Baustein kann wahlweise mit (Voreinstellung) oder ohne Eingang betrieben werden. In der Betriebsart mit Eingang wird der anliegende Eingangswert mit der *Verstärkung* des Schiebereglers multipliziert ausgegeben. In der Betriebsart ohne Eingang liegt der Ausgangswert je nach Potistellung zwischen 0 und der vorgegebenen Verstärkung. Zusätzlich kann eine Digitalanzeige der auf die Potistellung bezogenen Verstärkung als Absolut- oder Realtivwert (in %) erfolgen.

Steuerfenster des Schiebereglers mit absoluter bzw. relativer Digitalanzeige

Parameter-dialog:



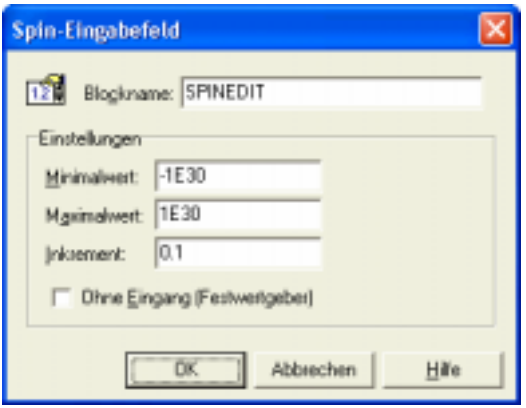
Spin-Eingabefeld

Typname: SPINEDIT

Funktion: Dieser Block realisiert ein numerisches Eingabefeld mit Wippregler (Spin-Element). Minimal-, Maximalwert und Inkrement des Elements sind einstellbar. Das Element kann wahlweise als Verstärker (mit Blockeingang) sowie als Festwertgeber (ohne Blockeingang) eingesetzt werden.

Steuerfenster des Spin-Eingabefelds

Parameter-dialog:



Drehregler

Typname: DREHREGLER

Funktion: Dieser Block realisiert einen vom Anwender mit der linken Maustaste zu betätigenden Drehregler. Der Baustein kann wahlweise mit oder ohne Eingang betrieben werden. In der Betriebsart *mit Eingang* wird der anliegende Eingangswert mit der aktuellen Einstellung des Drehreglers multipliziert ausgegeben. In der Betriebsart *ohne Eingang* wird der aktuell eingestellte Skalenwert direkt ausgegeben.

Steuerfenster des Drehreglers

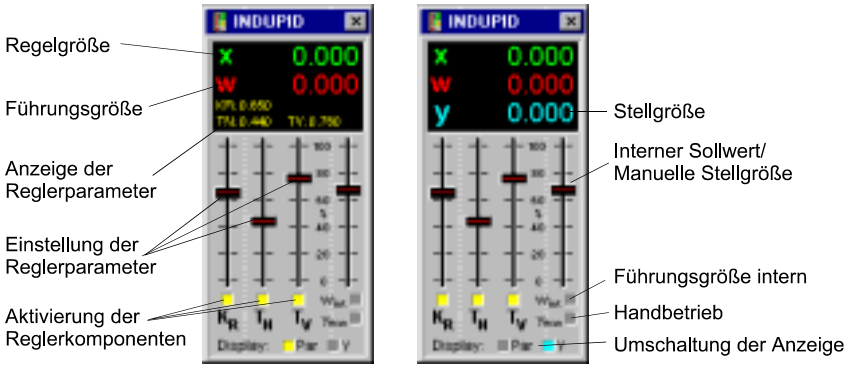
Parameter-dialog:

Industrie-PID-Regler

Typname: INDUPID

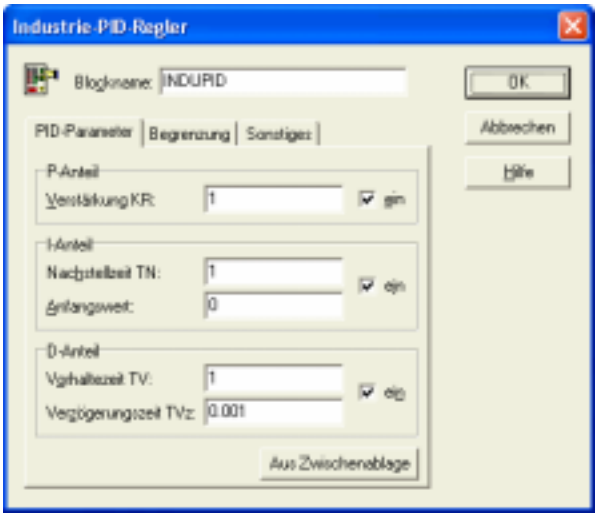
Funktion: Dieser Block realisiert einen PID-Regler mit einer Steuer- und Visualisierungsoberfläche, die einem Industrieregler nachempfunden wurde. Von der Funktionalität her entspricht dieser Blocktyp dem Standard-PID-Regler von BORIS, weist aber Soll- und Istwert (Führungsgröße *w* bzw. Regelgröße *x*) als getrennte Eingänge auf. Außerdem kann der Sollwert auf Wunsch auch intern generiert werden; der erste Eingang des Blocks ist dann ohne Bedeutung. Bei Bedarf kann der Regler auch im Handbetrieb (manuelle Vorgabe der Stellgröße) gefahren werden.

Die Reglerparameter sowie der interne Sollwert (falls aktiviert) können über Schieberegler variiert werden. Die einzelnen Regleranteile (P-, I- und D-Anteil) sind über Mausklick zu- bzw. abschaltbar. Im Display des Reglers werden neben Soll- und Istwert auch die aktuellen Reglerparameter bzw. wahlweise die aktuelle Stellgröße *y* angezeigt.



Steuer- und Visualisierungsfenster des Industrie-PID-Reglers mit Anzeige der Reglerparameter (links) bzw. der Stellgröße (rechts)

Parameterdialog:

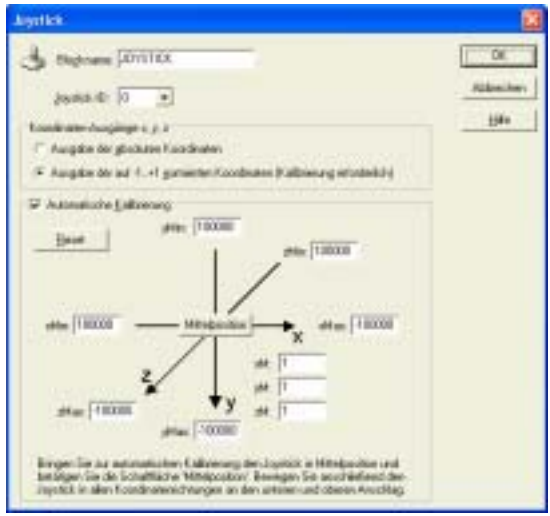


Joystick

Typname: JOYSTICK

Funktion: Dieser Block ermöglicht die Nutzung von bis zu zwei Joysticks am PC-Gameport. Die Blockausgänge *x*, *y* und *z* enthalten die Koordinaten (absolut oder normiert), die Ausgänge *B1* ... *B4* den Zustand der bis zu vier Feuerknöpfe des Joysticks als LOW- oder HIGH-Pegel. Über den Parameterdialog kann der Joystick automatisch kalibriert werden. Über die *Joystick-Id* (0 bzw. 1) wird der Joystick-Port ausgewählt.

Parameter-dialog:

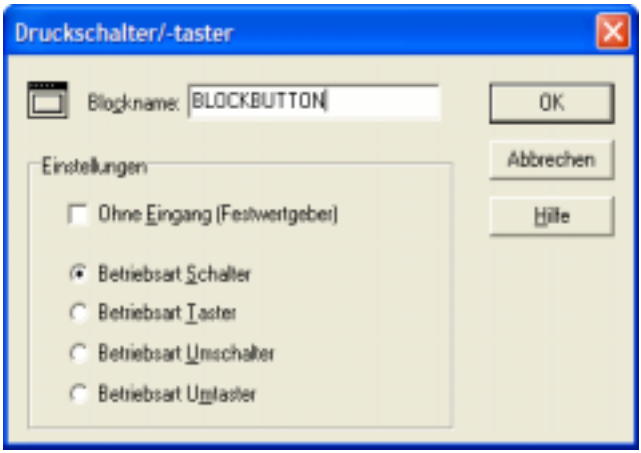


Block-Druckschalter

Typname: BLOCKBUTTON

Funktion: Dieser Block realisiert einen vom Anwender mit der linken Maustaste zu betätigenden Druckschalter bzw. -taster. Im Gegensatz zum normalen Druckschalter (Blocktyp BUTTON) erfolgt die Steuerung bei diesem Blocktyp jedoch nicht über ein separates Kontrollfenster, sondern die Schaltfläche befindet sich direkt innerhalb des Blockes selbst.

Parameter-dialog:



In der Betriebsart *Festwertgeber* wird am Blockausgang abhängig von der Schalterstellung ein logischer LOW- bzw. HIGH-Pegel ausgegeben.

Block-Schiebepotentiometer

Typname: BLOCKPOTI

Funktion: Dieser Block realisiert ein vom Anwender mit der linken Maustaste zu betätigendes Schiebepotentiometer. Im Gegensatz zum normalen Schiebepotentiometer (Blocktyp POTI) erfolgt die Steuerung bei diesem Blocktyp jedoch nicht über ein separates Kontrollfenster, sondern das Steuerelement befindet sich direkt innerhalb des Blockes selbst.

Parameter-dialog:

In der Betriebsart *Festwertgeber* wird am Blockausgang direkt der eingestellte Wert ausgegeben, andernfalls erhält der Blockausgang den mit dem eingestellten Wert multiplizierten Blockeingangswert.

Block-Spin-Eingabefeld

Typname: BLOCKSPINEDIT

Funktion: Dieser Block realisiert ein vom Anwender mit der linken Maustaste zu betätigendes Spin-Eingabefeld. Im Gegensatz zum normalen Spin-Eingabefeld (Blocktyp SPINEDIT) erfolgt die Steuerung bei diesem Blocktyp jedoch nicht über ein separates Kontrollfenster, sondern das Steuerelement befindet sich direkt innerhalb des Blockes selbst.

Parameterdialog:

In der Betriebsart *Festwertgeber* wird am Blockausgang direkt der eingestellte Wert ausgegeben, andernfalls erhält der Blockausgang den mit dem eingestellten Wert multiplizierten Blockeingangswert.

Kommunikation

DDE-Eingang

Typname: DDEIN

Funktion: Dieser Block ermöglicht das Einlesen von Werten aus anderen Windows-Anwendungen (z. B. EXCEL oder LabView) über *Dynamic Data Exchange* (DDE).

Der Datenlieferant (*Server*) wird über folgende Parameter spezifiziert:

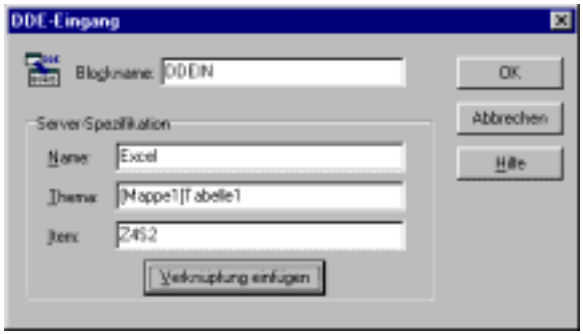
Name Name der Server-Applikation (z. B. EXCEL)

Thema Thema der DDE-Konversation (z. B. Name der EXCEL-Tabelle, also z. B. TAB1)

Item Spezifizierung des einzulesenden Wertes (z. B. Z1S1 für die erste Zeile, erste Spalte der EXCEL-Tabelle)

Bei der Spezifizierung des Items kann auch eine Zählvariable der Form *{#n}* eingefügt werden, um z. B. alle Werte fortlaufend aus einer Spalte einer EXCEL-Tabelle zu lesen. Soll beispielsweise die erste Spalte gelesen werden, wobei der erste Wert aus der ersten Zeile, der zweite aus der zweiten Zeile usw. gelesen wird, so lautet der Eintrag *Z{#1}S1*. Soll erst in der zweiten Zeile begonnen werden, lautet der Eintrag dementsprechend *Z{#2}S1*. Sollen die Werte hingegen z. B. zeilenweise aus der ersten Zeile gelesen werden, lautet der Eintrag *Z1S{#1}* usw. Man beachte hierbei, dass der erste Wert grundsätzlich bei der Initialisierung der Simulation gelesen wird, d. h. im ersten Simulationsschritt bereits der zweite Wert gelesen wird usw.

Parameterdialog:



Über die Schaltfläche *Verknüpfung einfügen* kann eine zuvor von einer anderen Anwendung in die Zwischenablage kopierte Verknüpfung automatisch eingefügt werden (in obigem Dialog z. B. ein Verweis auf eine Zelle eines Excel-Arbeitsblattes).

Hinweis: Im Lieferumfang von WinFACT befindet sich ein Demo-Programm mit Namen DDEDEMO.EXE, mit dem Sie die DDE-

Fähigkeiten von BORIS ausprobieren können. Starten Sie das Programm, starten Sie dann BORIS und laden Sie dort die Beispieldatei DDEDEMO.BSY. Danach starten Sie die Simulation. Sie können nun in DDEDEMO Daten editieren, die dann in BORIS angezeigt werden; umgekehrt werden die von BORIS generierten Daten in DDEDEMO angezeigt!

DDE-Ausgang

Typname: DDEOUT

Funktion: Dieser Block ermöglicht die Ausgabe von Werten zu anderen Windows-Anwendungen (z. B. EXCEL oder LabView) über *Dynamic Data Exchange* (DDE).

Der Datenempfänger (*Server*) wird über folgende Parameter spezifiziert:

Name Name der Server-Applikation (z. B. EXCEL)

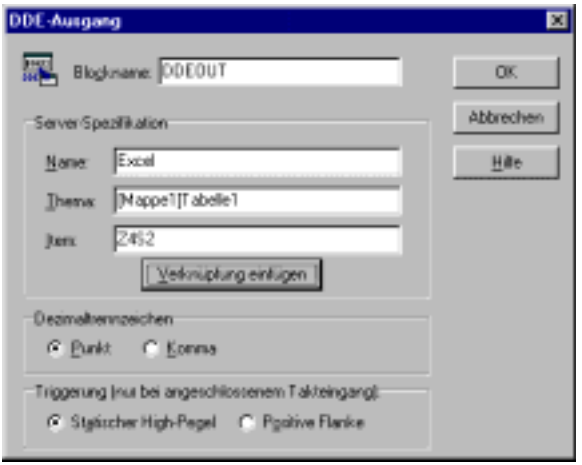
Thema Thema der DDE-Konversation (z. B. Name der EXCEL-Tabelle, also z. B. TAB1)

Item Spezifizierung des Ziels für den ausgegebenen Wert (z. B. Z1S1 für die erste Zeile, erste Spalte der EXCEL-Tabelle)

Bei der Spezifizierung des Items kann auch eine Zählvariable der Form *{#n}* eingefügt werden, um z. B. alle Werte fortlaufend in eine Spalte einer EXCEL-Tabelle zu schreiben. Soll beispielsweise die erste Spalte beschrieben werden, wobei der erste Wert in die erste Zeile, der zweite in die zweite Zeile usw. geschrieben wird, so lautet der Eintrag *Z{#1}S1*. Soll erst in der zweiten Zeile begonnen werden, lautet der Eintrag dementsprechend *Z{#2}S1*. Sollen die Werte hingegen z. B. zeilenweise in die erste Zeile geschrieben werden, lautet der Eintrag *Z1S{#1}* usw. Man beachte hierbei, dass der erste Wert grundsätzlich bei der Initialisierung der Simulation geschrieben wird, d. h. im ersten Simulationsschritt bereits der zweite Wert geschrieben wird usw.

Bei angeschlossenem Triggereingang C erfolgt eine Ausgabe wahlweise nur bei einer positiven Flanke am Triggereingang oder bei statischem High-Pegel. Das Dezimaltrennzeichen (Punkt bzw. Komma) kann über die entsprechenden Schalter frei gewählt werden.

Parameter-dialog:



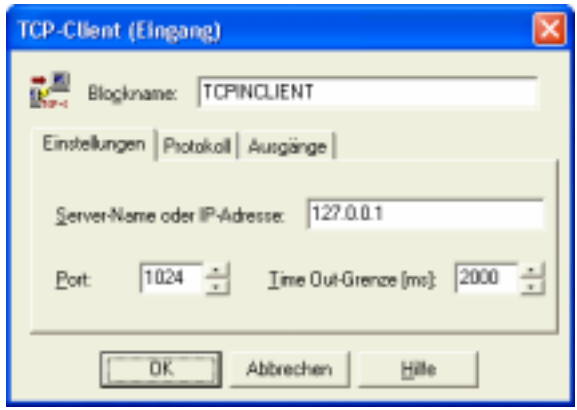
Über die Schaltfläche *Verknüpfung einfügen* kann eine zuvor von einer anderen Anwendung in die Zwischenablage kopierte Verknüpfung automatisch eingefügt werden (in obigem Dialog z. B. ein Verweis auf eine Zelle eines Excel-Arbeitsblattes).

TCP-Client (Eingang)

Typname: TCPINCLIENT

Funktion: Dieser Eingangsblock realisiert einen Client, der Daten nach dem TCP/IP-Protokoll empfängt. Der aktuelle Status der Verbindung wird über ein separates Statusfenster angezeigt. Die genaue Funktionalität erläutert die Datei TCPDEMO.BSY bzw. die Dateien TCPCLIENT.BSY/TCPSERVER.BSY (beide Dateien in getrennte BORIS-Instanzen laden und dann beide Simulationen starten!) aus dem Examples-Verzeichnis. Bei mehr als einem Blockausgang sind alle Kanäle in einem einzigen String zu übertragen, wobei die einzelnen Kanäle durch Semikola zu trennen sind.

Parameter-dialog:



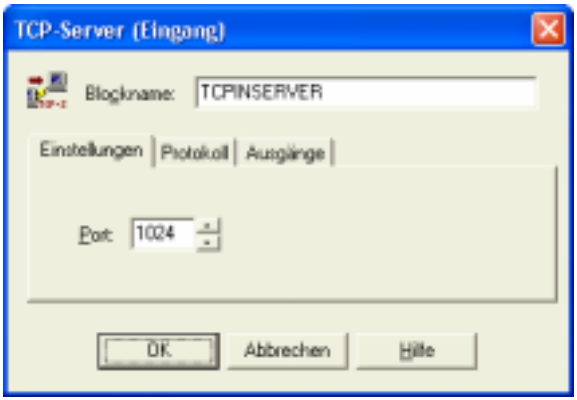
Ist die Option *Trigger-Eingang verwenden* aktiviert, so besitzt der Block einen Trigger-Eingang. In diesem Fall findet eine Übertragung der Daten nur statt, solange der Trigger-Eingang HIGH-Pegel aufweist.

TCP-Server (Eingang)

Typname: TCPINSERVER

Funktion: Dieser Eingangsblock realisiert einen Server, der Daten nach dem TCP/IP-Protokoll empfängt. Der aktuelle Status der Verbindung wird über ein separates Statusfenster angezeigt. Die genaue Funktionalität erläutert die Datei TCPDEMO.BSY bzw. die Dateien TCPCLIENT.BSY/TCPSERVER.BSY (beide Dateien in getrennte BORIS-Instanzen laden und dann beide Simulationen starten!) aus dem Examples-Verzeichnis. Bei mehr als einem Blockausgang sind alle Kanäle in einem einzigen String zu übertragen, wobei die einzelnen Kanäle durch Semikola zu trennen sind.

Parameter-dialog:

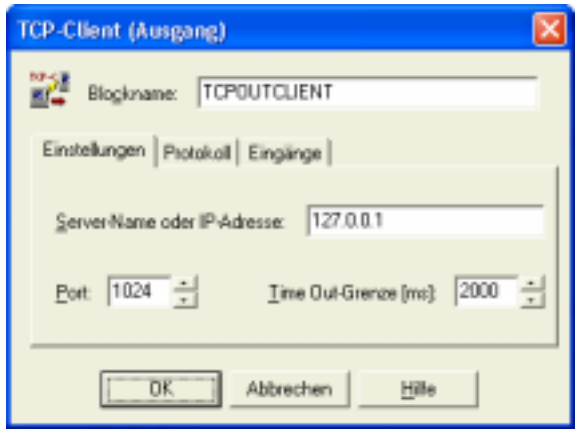


TCP-Client (Ausgang)

Typname: TCPOUTCLIENT

Funktion: Dieser Ausgangsblock realisiert einen Client, der Daten nach dem TCP/IP-Protokoll versendet. Der aktuelle Status der Verbindung wird über ein separates Statusfenster angezeigt. Die genaue Funktionalität erläutert die Datei TCPDEMO.BSY bzw. die Dateien TCPCLIENT.BSY/TCPSERVER.BSY (beide Dateien in getrennte BORIS-Instanzen laden und dann beide Simulationen starten!) aus dem Examples-Verzeichnis. Bei mehr als einem Blockeingang sind alle Kanäle in einem einzigen String zu übertragen, wobei die einzelnen Kanäle durch Semikola zu trennen sind.

Parameter-dialog:



Ist die Option *Letzten Eingang als Trigger-Eingang verwenden* aktiviert, so wird der letzte Blockeingang als Trigger-Eingang benutzt. In diesem Fall findet eine Übertragung der Daten nur statt, solange der Trigger-Eingang HIGH-Pegel aufweist.

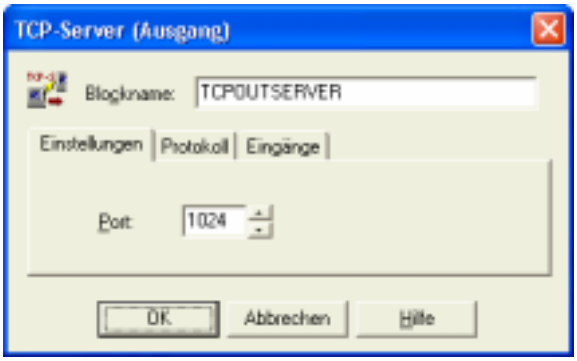
Ist die Option *Senden nur bei Wertänderung* aktiviert, so findet eine Übertragung der Daten nur statt, wenn sich der Wert mindestens eines Blockeingangs geändert hat.

TCP-Server (Ausgang)

Typname: TCPOUTSERVER

Funktion: Dieser Ausgangsblock realisiert einen Server, der Daten nach dem TCP/IP-Protokoll versendet. Der aktuelle Status der Verbindung wird über ein separates Statusfenster angezeigt. Die genaue Funktionalität erläutert die Datei TCPDEMO.BSY bzw. die Dateien TCPCLIENT.BSY/TCPSERVER.BSY (beide Dateien in getrennte BORIS-Instanzen laden und dann beide Simulationen starten!) aus dem Examples-Verzeichnis. Bei mehr als einem Blockeingang sind alle Kanäle in einem einzigen String zu übertragen, wobei die einzelnen Kanäle durch Semikola zu trennen sind.

Parameter-dialog:



E-Mail-Client

Typname: EMAIL

Funktion: Dieser Block versendet bei bestehender Online-Verbindung eine E-Mail (auf Wunsch mit Anhang), sobald am Blockeingang eine positive Flanke auftritt. Der Verbindungsaufbau und das Versenden der E-Mail werden in einem separaten Statusfenster protokolliert.

Parameter-dialog:

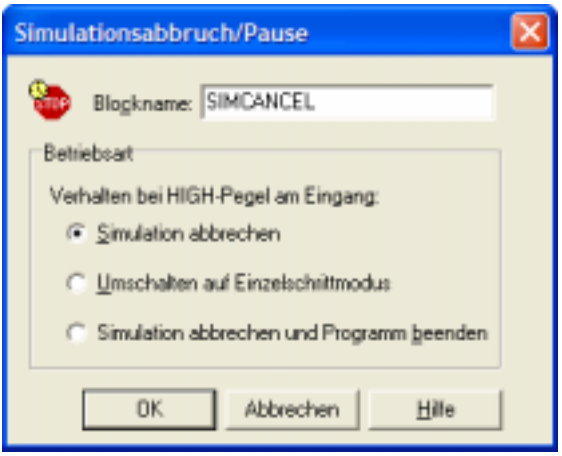
Simulationssteuerung

Simulationsabbruch/Pause

Typname: SIMCANCEL

Funktion: Dieser Block bricht die Simulation ab (und beendet auf Wunsch auch das Programm) bzw. schaltet in den Einzelschrittmodus, sobald an seinem Eingang High-Pegel anliegt.

Parameter-dialog:

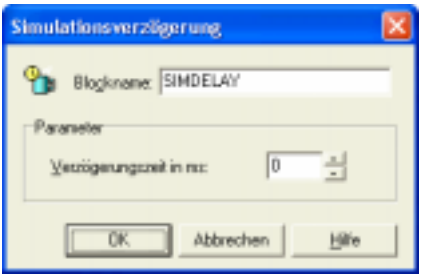


Simulationsverzögerung

Typname: SIMDELAY

Funktion: Dieser Block verzögert die Simulation bei jedem Simulationsschritt um eine vorgebbare Zeit in Millisekunden. Er kann benutzt werden, um auf schnellen Rechnern eine künstliche Verlangsamung des Simulationsablaufs zu erreichen.

Parameter-dialog:

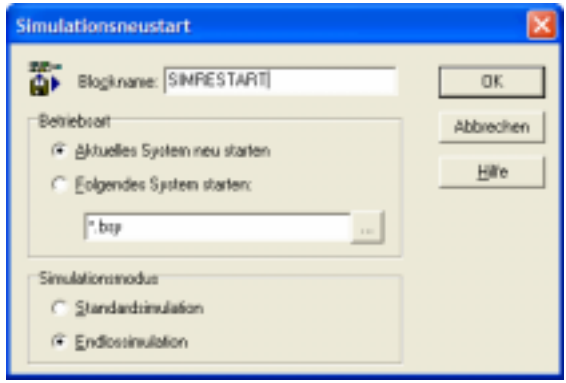


Simulationsneustart

Typname: SIMRESTART

Funktion: Dieser Block bricht die aktuell laufende Simulation ab und startet eine neue Simulation, sobald an seinem Eingang eine positive Flanke auftritt. Die neu gestartete Simulation kann auch auf einer anderen Systemdatei (BSY-Datei) als der aktuell geladenen basieren und wahlweise als Standard- oder Endlossimulation gestartet werden.

Parameterdialog:



Zustands-Ausgabedatei

Typname: STATE2FILE

Funktion: Dieser Block ermöglicht die Ausgabe des Systemzustands in eine BSF-Datei. Einzelheiten dazu entnehmen Sie bitte dem Abschnitt *Speichern und Laden von Systemzuständen*.

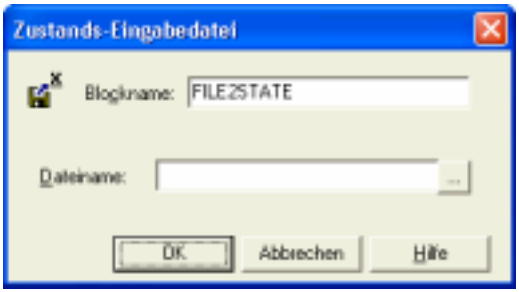
Parameterdialog:

Zustands-Eingabedatei

Typname: FILE2STATE

Funktion: Dieser Block ermöglicht das Einlesen des Systemzustands aus einer BSF-Datei. Einzelheiten dazu entnehmen Sie bitte dem Abschnitt *Speichern und Laden von Systemzuständen*.

Parameter-dialog:



Ausgangsblöcke (Senken)

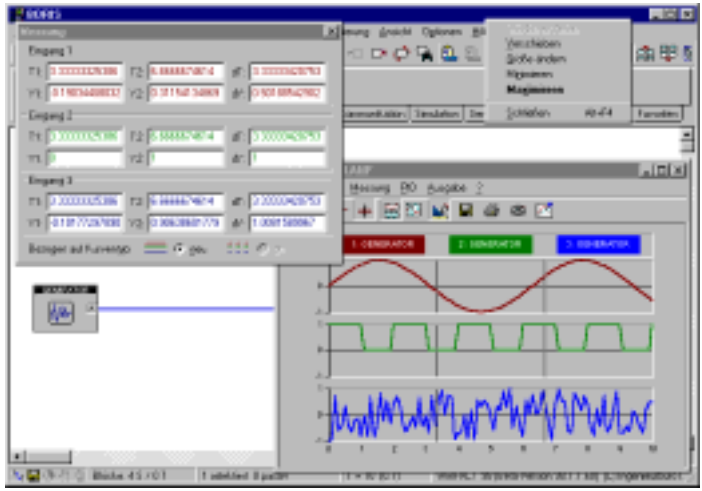
Zeitverlauf

Typname: ZEITVERLAUF

Funktion: Dieser Ausgangsblock ermöglicht die gleichzeitige Darstellung von bis zu drei Zeitverläufen in einem beliebig vergrößerbaren Anzeigefenster. Die Kurven können in ein gemeinsames oder in getrennte Diagramme, mit und ohne Raster gezeichnet werden. Die Skalierung der Koordinatenachsen kann manuell oder automatisch erfolgen. Die Amplitudenwerte können sowohl linear als auch logarithmisch aufgetragen werden. Außerdem ist ein direktes Abspeichern von Kurven in SIM-Dateien möglich.

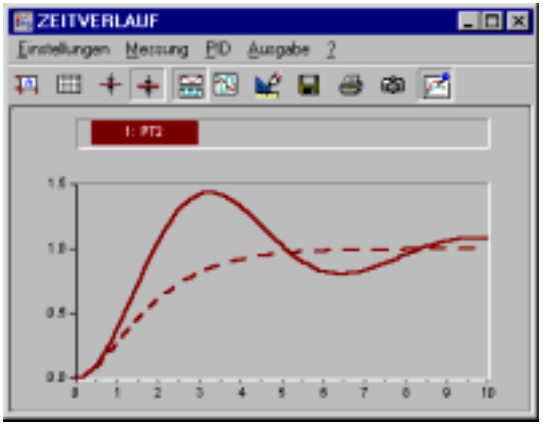
Ist der Reset-Eingang R des Blocks angeschlossen, so wird die Anzeige über eine positive Flanke an diesem Eingang zurückgesetzt; die Aufzeichnung des Blocks startet also ab diesem Zeitpunkt neu.

Über eine komfortable Messfunktion können den Diagrammen auf einfache Weise Messwerte entnommen werden. Da die Zeitverläufe der vorangegangenen Simulation jeweils "eingefroren" werden können, sind auch Trends bei Parametervariationen - z. B. von Reglerparametern - darstellbar. Die Messfunktion kann bei Bedarf auch auf die eingefrorenen Kurven angewendet werden. Speziell für regelungstechnische Anwendungen (z. B. das Ablesen von Ausregelzeiten) kann zusätzlich ein *Toleranzband* mit frei wählbarer Breite in die Diagramme eingeblendet werden.



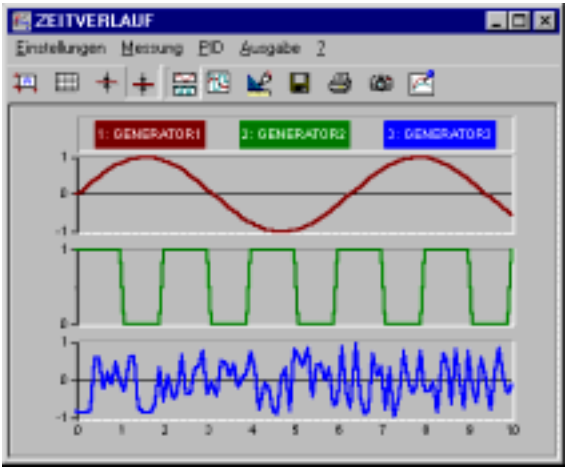
Bildschirm bei aktivierter Messfunktion

Weiterhin ist aus dem Anzeigefenster heraus der direkte Entwurf von PID-Reglern nach Einstellregeln möglich. Auf diese Funktion wird an anderer Stelle im Kapitel *Entwurf von PID-Reglern* detailliert eingegangen.



Zeitverlauffenster bei aktivierter Speicherfunktion: Die im vorangegangenen Simulationslauf ermittelte Kurve wird gestrichelt eingezeichnet

Anzeige-fenster:



Oberhalb der Diagramme werden die Bezeichner der zugehörigen Systemblöcke (hier *Generator1*, *Generator2* und *Generator3*) angegeben. Die Toolbar unterhalb des Fenstermenüs erlaubt den Direktzugriff auf die wichtigsten Menüoptionen bzw. Einstellungen des Parameterdialogs, der im übrigen auch über die Menüoption EINSTELLUNGEN erreicht werden kann.

Hinweis: Wird beim Einfügen des Fensterinhalts in die Zwischenablage über das Kamera-Symbol der Toolbar zusätzlich eine der beiden <Shift>-Tasten gedrückt, so wird der graue Fensterhintergrund durch einen weißen ersetzt.

Dies kann z. B. sinnvoll sein, wenn der Screenshot in ein Textdokument eingebunden und ausgedruckt werden soll.

Parameterdialog:

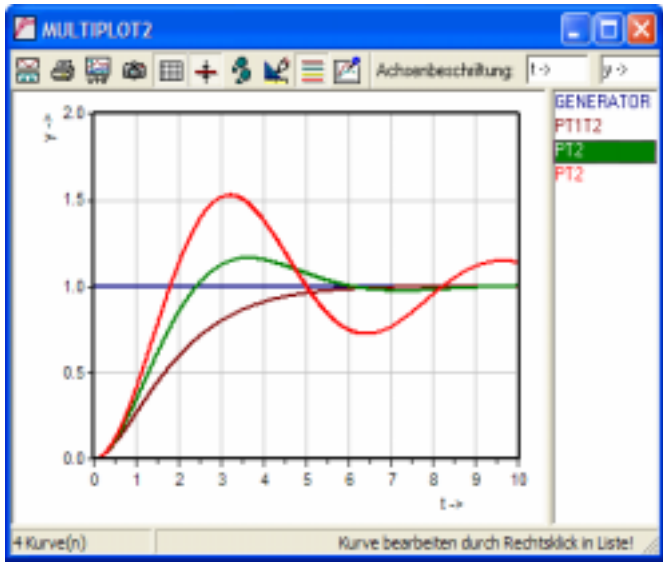
Parametergrenzen: Für jede Kurve stehen intern maximal 32000 Punkte zur Verfügung. Ist die Anzahl der Simulationsschritte größer, so werden Zwischenwerte übersprungen ("komprimierte" Darstellung). Beträgt die Anzahl der Simulationsschritte beispielsweise 64000, so wird nur jeder zweite Simulationsschritt grafisch dargestellt. Dies ist zu beachten, wenn etwa sehr kurze Impulse dargestellt werden sollen. Diese komprimierte Darstellung wird daher durch den Schriftzug *COMPR!* in rot auf gelbem Grund in der linken oberen Ecke des Anzeigefensters angezeigt.

Mehrfach-Zeitverlauf

Typname: MULTIPLOT2

Funktion: Dieser Anzeigeblock ermöglicht die parallele Aufzeichnung von bis zu 50 Signalen, die in einem gemeinsamen oder separaten Koordinatensystemen dargestellt werden können.

Anzeige-fenster:



Über die Toolbar am oberen Fensterrand steht eine Vielzahl von weiteren Optionen zur Verfügung, die im Wesentlichen selbsterklärend sind. Durch einen Rechtsklick innerhalb der Liste am rechten Fensterrand können einzelne Kurven bearbeitet (z. B. skaliert) oder auch gespeichert werden. Soll eine manuelle Skalierung in dem Fall erfolgen, dass alle Kurven in einem gemeinsamen Diagramm dargestellt werden, so werden die für die *erste* Kurve (d. h. den obersten Listeneintrag) angegebenen Minimal- und Maximalwerte benutzt.

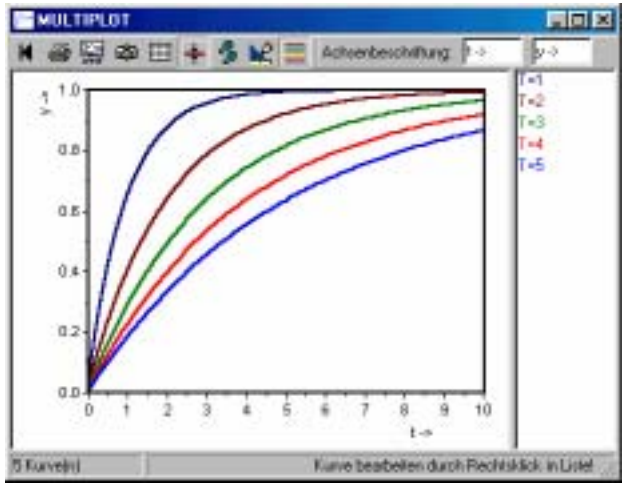
Parameter-dialog:

Mehrfach-Plotter

Typname: MULTIPLOT

Funktion: Dieser Anzeigeblock ermöglicht die Aufzeichnung von Kurvenverläufen über mehrere Simulationen und ist damit insbesondere für die Untersuchungen im Zusammenhang mit Parametervariationen (z. B. Zeitkonstanten) und Batchläufen interessant. Es können beliebig viele Kurven aufgezeichnet werden, die in einer Liste am rechten Fensterrand angezeigt werden. Durch einen Rechtsklick auf einen Listeneintrag steht ein Kontextmenü mit zusätzlichen Optionen zur Kurvenverwaltung zur Verfügung. Im Rahmen von Batchläufen (siehe Kapitel *Simulationen im Batch-Betrieb*) erfolgt eine automatische Betitelung der Kurven in Abhängigkeit von den variierten Blockparametern.

Anzeigefenster:



Über die Toolbar am oberen Fensterrand steht eine Vielzahl von weiteren Optionen zur Verfügung, die im Wesentlichen selbsterklärend sind. Wichtig ist dabei insbesondere die Schaltfläche  , über die ein Reset des Mehrfach-Plotters (d. h. ein Löschen aller zuvor aufgezeichneten Kurven) erfolgt.

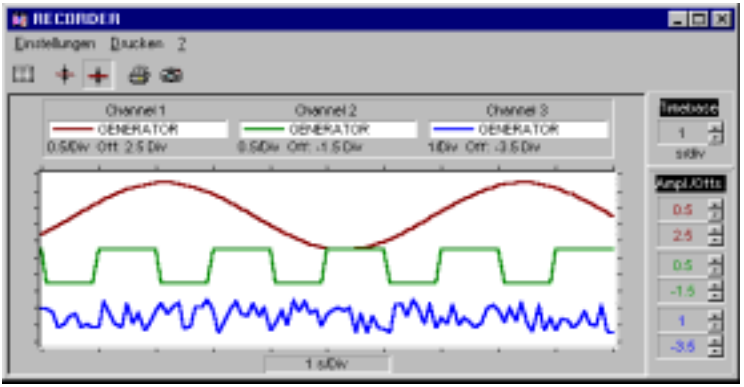
Parameterdialog:

y-t-Schreiber (Recorder)

Typname: RECORDER

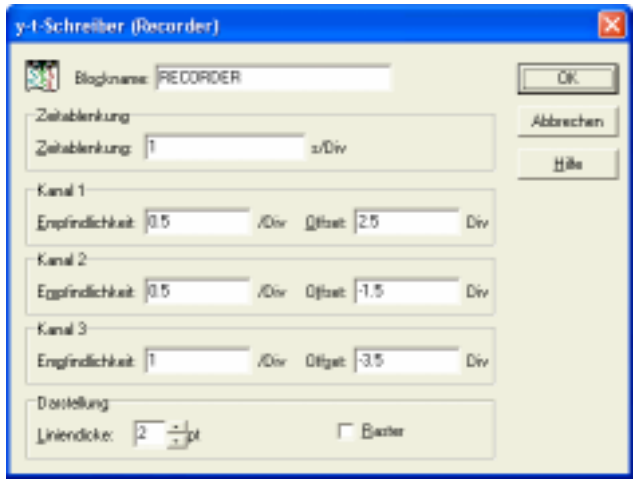
Funktion: Dieser Block bildet einen y-t-Schreiber nach und ist damit insbesondere für Langzeitaufzeichnungen geeignet. Der Block kann bis zu drei Eingangskanäle verarbeiten. Zeitbasis sowie Ablenkempfindlichkeit und Offset der einzelnen Kanäle sind frei wählbar.

Anzeige-fenster:



Oberhalb der Diagramme werden die Bezeichner der zugehörigen Systemblöcke (hier *Generator1*, *Generator2* und *Generator3*) angegeben. Die Toolbar unterhalb des Fenstermenüs erlaubt den Direktzugriff auf die wichtigsten Menüoptionen bzw. Einstellungen des Parameterdialogs, der im übrigen auch über die Menüoption EINSTELLUNGEN erreicht werden kann.

Parameter-dialog:

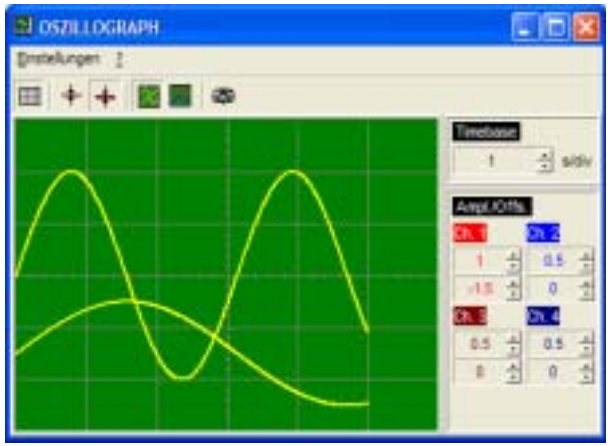


Oszillograph

Typname: OSZILLOGRAPH

Funktion: Dieser Block bildet einen Vierkanal-Oszillographen auf dem Bildschirm nach, der beliebig verkleinert und vergrößert werden kann. Die Ablenkempfindlichkeit und der Nullpunkt sind für alle Kanäle getrennt einstellbar. Ebenso ist die Zeitablenkung variierbar. Die Kanäle können auf Wunsch mehrfarbig dargestellt werden.

Anzeige-fenster:



Die eingestellten Empfindlichkeiten werden in der rechten oberen Ecke angegeben. Die Toolbar unterhalb des Fenstermenüs erlaubt den Direktzugriff auf die wichtigsten Einstellungen des Parameterdialogs, der im übrigen auch über die Menüoption EINSTELLUNGEN erreicht werden kann.

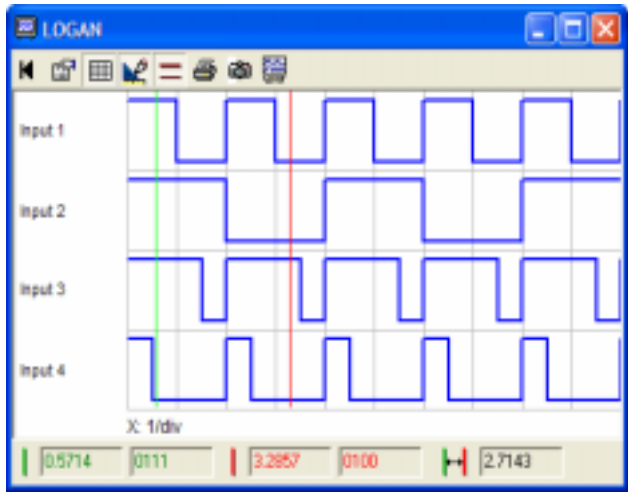
Parameter-grenzen: Für jede Kurve stehen intern maximal 32000 Punkte zur Verfügung. Ist die Zahl der Simulationsschritte, die auf die Breite des Oszillographs entfallen, größer, so werden Zwischenwerte übersprungen ("komprimierte" Darstellung). Beträgt die Anzahl der Schritte beispielsweise 64000, so wird nur jeder zweite Simulationsschritt grafisch dargestellt. Dies ist zu beachten, wenn etwa sehr kurze Impulse dargestellt werden sollen. Diese komprimierte Darstellung wird daher durch den Schriftzug *COMPR!* in rot auf gelbem Grund in der rechten unteren Ecke des Anzeigefensters angezeigt (vgl. Zeitverlauf-Block!)

Logik-Analysator

Typname: LOGAN

Funktion: Dieser Block stellt einen Logik-Analysator mit bis zu 49 Dateneingängen und einem Reset-Eingang R dar. Der Logik-Analysator besitzt u. a. eine komfortable Messfunktion.

Anzeige-fenster:



Über die Toolbar am oberen Fensterrand sind die verschiedenen Optionen des Logik-Analysators zugänglich:

Aktiviert die Auto-Reset-Funktion. Ist diese aktiv, so wird die Anzeige beim Erreichen des rechtes Fensterrandes automatisch gelöscht und mit der Neuaufzeichnung der Kurven begonnen.

Ermöglicht die Vorgabe von Zeitbasis und Zeiteinheiten sowie die Benennung der Eingangsgrößen.

Schaltet das Koordinatenraster ein bzw. aus.

Schaltet die Messfunktion ein bzw. aus.

Schaltet die Linienbreite der Kurven um.

Ausdrucken des Fensterinhalts

Kopieren des Fensterinhalts in die Zwischenablage

Speichern des Fensterinhalts in einer WMF-Datei

Analoganzeige

Typname: ANALOGANZEIGE

Funktion: Dieser Block stellt die Nachbildung eines Analoginstruments - auf Wunsch mit Digitalanzeige - dar. Bestimmte Skalenbereiche können rot oder grün dargestellt und die Anzeige mit einer Einheit (z. B. "V") versehen werden.

Anzeigefenster:

Parameterdialog:

Für den Skalenbereich $y_{\min}$, $y_{\max}$ sollten möglichst ganzzahlige Werte vorgegeben werden.

Digitalanzeige

Typname: DIGITALANZEIGE

Funktion: Dieser Block bildet ein Digitalinstrument nach. Das zugehörige Anzeigefenster besitzt eine feste Größe. Auf Wunsch kann ein zusätzlicher Skalenfaktor und eine Einheit (z. B. "V") sowie eine Trendanzeige (Eingangswert steigend/fallend) vorgegeben werden. Die Ausgabe des numerischen Werts erfolgt wahlweise mit möglichst geringer oder fest vorgegebener Stellenzahl.

Anzeige-fenster:

Parameter-dialog:

Balkendiagramm

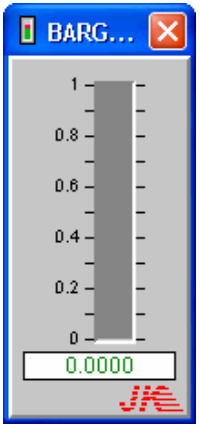
Typname: BARGRAPH

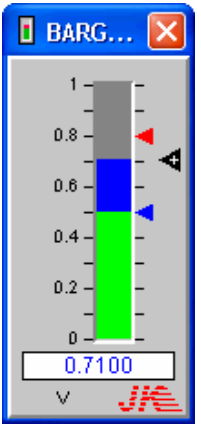
Funktion: Dieser Ausgabeblock stellt ein Balkendiagramm (Bargraph) dar, das in verschiedenen Modi betrieben werden kann. Im einzelnen sind folgende Optionen möglich:

- Setzen von ein oder zwei Markern, bei deren Überschreitung die Anzeige ihre Farbe wechselt.

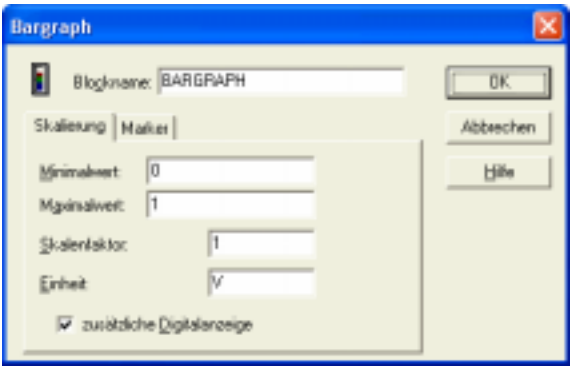
- Setzen von Markern für minimal und maximal erreichten Wert.
- Zusätzliche Digitalanzeige, Skalenfaktor und Einheit (z. B. "V")

Anzeige-fenster:





Parameter-dialog:



Vertikale LED-Anzeige

Typname: VLEDBAR

Funktion: Dieser Ausgabeblock stellt eine mehrfarbige vertikale LED-Anzeige dar, die auf Wunsch mit einer Maximalwertspeicherung versehen werden kann.

Anzeige-fenster:

Parameter-dialog:

Horizontale LED-Anzeige

Typname: HLEDBAR

Funktion: Dieser Ausgabeblock stellt eine mehrfarbige horizontale LED-Anzeige dar, die auf Wunsch mit einer Maximalwertspeicherung versehen werden kann.

Anzeige-fenster:

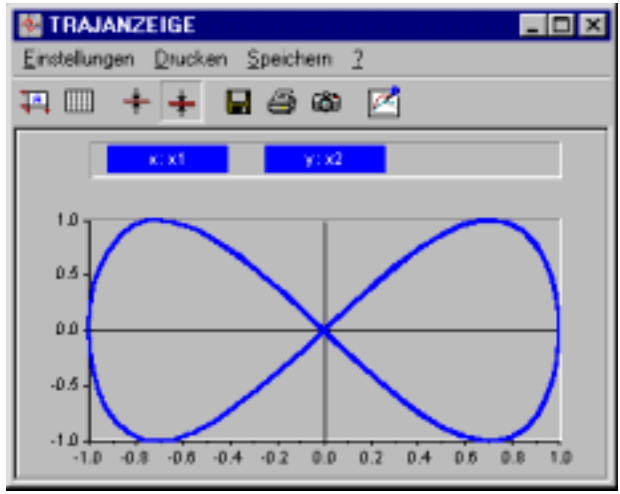
Parameter-dialog:

Trajektorienanzeige

Typname: TRAJANZEIGE

Funktion: Die Trajektorienanzeige erlaubt die Darstellung zweier Eingangsgrößen $x(t)$ und $y(t)$ in der x-y-Ebene mit der Zeit t als Kurvenparameter. Das zugehörige Anzeigefenster kann beliebig verkleinert und vergrößert werden. Die Skalierung der Koordinatenachsen kann manuell oder automatisch, mit oder ohne Raster erfolgen. Wie beim Zeitverlauf-Block steht auch hier eine Speicherfunktion für die vorangegangene Simulation zur Verfügung.

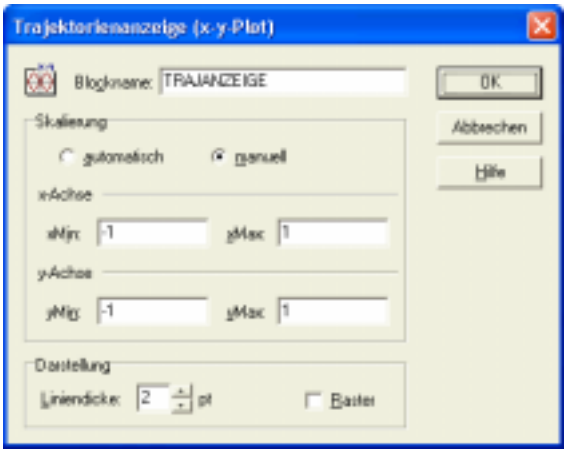
Anzeige-fenster:



Oberhalb des Diagramms werden die Bezeichner der zugehörigen Systemblöcke (hier *x1* und *x2*) angegeben. Die Toolbar unterhalb des Fenstermenüs erlaubt den Direktzugriff auf die wichtigsten Einstellungen des Parameterdialogs, der im übrigen auch über die Menüoption EINSTELLUNGEN erreicht werden kann.

Hinweis: Wird beim Einfügen des Fensterinhalts in die Zwischenablage über das Kamera-Symbol der Toolbar zusätzlich eine der beiden <Shift>-Tasten gedrückt, so wird der graue Fensterhintergrund durch einen weißen ersetzt. Dies kann z. B. sinnvoll sein, wenn der Screenshot in ein Textdokument eingebunden und ausgedruckt werden soll.

Parameter-dialog:



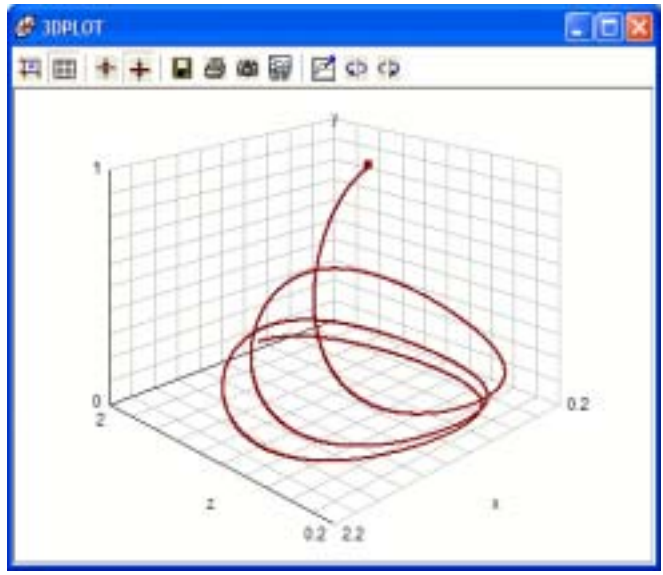
Parameter-grenzen: Für jede Kurve stehen intern maximal 32000 Punkte zur Verfügung. Ist die Anzahl der Simulationsschritte größer, so werden Zwischenwerte übersprungen ("komprimierte" Darstellung). Beträgt die Anzahl der Simulationsschritte beispielsweise 64000, so wird nur jeder zweite Simulationsschritt grafisch dargestellt. Dies ist zu beachten, wenn etwa sehr kurze Impulse dargestellt werden sollen. Diese komprimierte Darstellung wird daher durch den Schriftzug *COMPR!* in rot auf gelbem Grund in der rechten unteren Ecke des Anzeigefensters angezeigt (vgl. Zeitverlauf-Block!).

3D-Plotter

Typname: 3DPLOT

Funktion: Dieser Anzeigeblock erlaubt die Darstellung dreier Eingangsgrößen $x(t)$, $y(t)$ und $z(t)$ in der *x*-*y*-*z*-Ebene mit der Zeit *t* als Kurvenparameter. Die Grafik kann wahlweise manuell oder automatisch skaliert und frei im Raum gedreht werden.

Anzeige-fenster:



Über die Toolbar am oberen Fensterrand steht eine Vielzahl von Optionen zur Verfügung, die im Wesentlichen selbsterklärend sind.

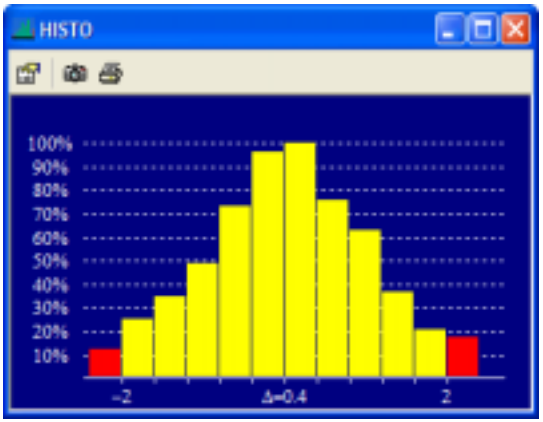
Parameter-dialog:

Histogramm

Typname: HISTO

Funktion: Dieser Anzeigeblock ermöglicht die Darstellung der Häufigkeitsverteilung der Werte des Eingangssignals. Der Darstellungsbereich und die Anzahl der Intervalle der Eingangsgröße sind einstellbar.

Anzeige-fenster:



Über die Toolbar am oberen Fensterrand steht eine Reihe von Optionen zur Verfügung, die im Wesentlichen selbsterklärend sind.

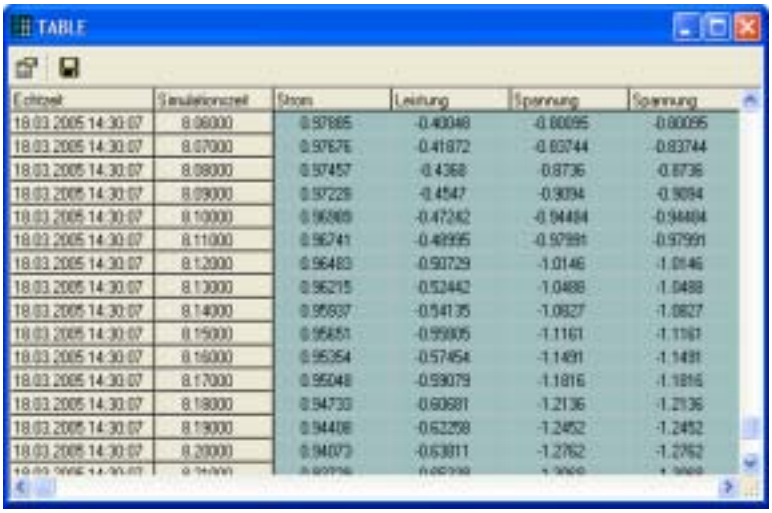
Parameter-dialog:

Tabelle

Typname: TABLE

Funktion: Dieser Anzeigeblock ermöglicht die tabellarische Ausgabe von bis zu 50 Eingangsgrößen. Das Ausgabeintervall ist frei wählbar; ebenso kann neben den Amplitudenwerten eine Zeitspalte mit der aktuellen Zeit und/oder Simulationszeit erstellt werden.

Anzeige-fenster:



Über die Toolbar am oberen Fensterrand steht eine Reihe von Optionen zur Verfügung, die im Wesentlichen selbsterklärend sind.

Parameter-dialog:

Wird für das *Ausgabeintervall* ein Wert von 0 (Voreinstellung) angegeben, erfolgt eine Ausgabe bei *jedem* Simulationsschritt.

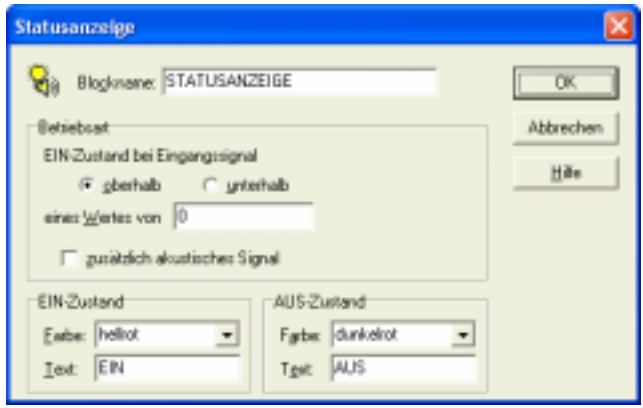
Statusanzeige

Typname: STATUSANZEIGE

Funktion: Dieser Block erzeugt eine visuelle und auf Wunsch eine zusätzliche akustische Kontrollausgabe beim Über- bzw. Unterschreiten eines vorgebbaren Schwellwerts. Der Text im Display ist für den EIN- und AUS-Zustand getrennt vorgebbar.

Anzeigefenster:

Parameterdialog:

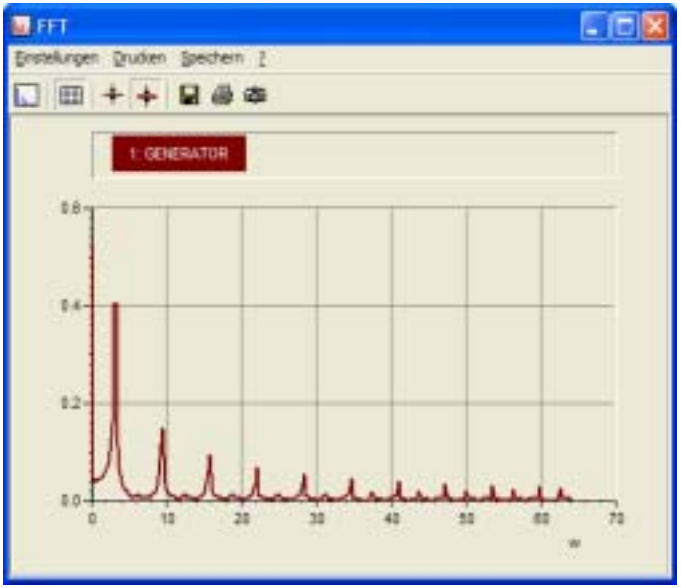


Fast-Fourier-Transformation (FFT)

Typname: FFT

Funktion: Dieser Block ermöglicht die Berechnung des Amplitudenspektrums des Eingangssignals über die Fast-Fourier-Transformation (FFT). Die Stützstellenzahl und das Zeitfenster für die Transformation sind wählbar. Der Block kann bis zu drei Eingänge besitzen. Zur Darstellung des Spektrums periodischer Signale kann der Block in die Betriebsart *diskret* versetzt werden; in diesem Fall werden jeweils die Maxima der ermittelten Spektrums in Form eines Linienspektrums angezeigt.

Anzeige-fenster:



Oberhalb des Diagramms wird der Bezeichner des zugehörigen Systemblocks (hier *Generator*) angegeben. Die Toolbar unterhalb des Fenstermenüs erlaubt den Direktzugriff auf die wichtigsten Einstellungen des Parameterdialogs, der im übrigen auch über die Menüoption EINSTELLUNGEN erreicht werden kann.

Parameter-dialog:

Die einzelnen Parameter haben folgende Bedeutung:

- Das *Zeitfenster T* gibt die Länge des Zeitfensters an, innerhalb dessen die Stützstellen für die Transformation liegen. Hierfür sollte in der Regel die Simulationsdauer gewählt werden; die Transformation erfolgt dann nach Beendigung der Simulation. Die Länge des Zeitfensters darf nicht größer als die Simulationsdauer gewählt werden, da ansonsten keine Transformation und damit auch keine Anzeige erfolgt.
- Das Zeitfenster legt die kleinste Frequenz $\omega_{\min}$ fest, für die das Spektrum berechnet wird. Es gilt

$$\omega_{\min} = \frac{2\pi}{T}.$$

- Die Anzahl der *Stützpunkte* ist immer eine Zweierpotenz und legt die obere Frequenz des berechneten Spektrums fest. Bei einer Zahl von *n* Stützstellen ergibt sich die obere Frequenz zu

$$\omega_{\max} = \left(\frac{n}{2} - 1\right)\omega_{\min}.$$

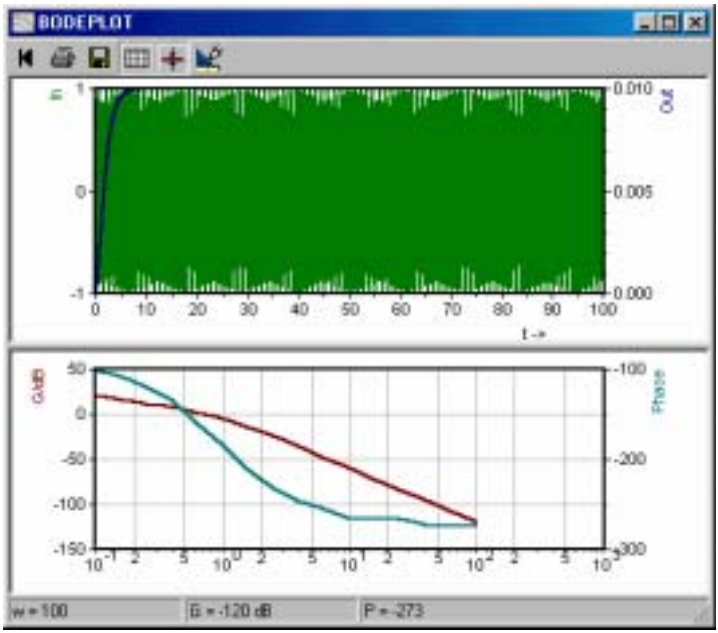
- Die Skalierung der Frequenzachse wird über die Einstellungen im Gruppenfeld *Frequenzachse* beeinflusst. Bei der automatischen Skalierung wird der gesamte Bereich bis $\omega_{\max}$ dargestellt.
- Über das Gruppenfeld *Anzeige* kann die Liniendicke der Kurve eingestellt sowie ein Koordinatenraster zugeschaltet werden.

Frequenzgang-Plotter

Typname: BODEPLOT

Funktion: Dieser Block ermöglich die Ermittlung von Betrag (in dB) und Phase (in Grad) zweier sinusförmiger Signale, die an den beiden Blockeingängen einzuspeisen sind und beide dieselbe Frequenz besitzen müssen. In Kombination mit dem Batch-Betrieb lässt sich der Block zur automatischen Berechnung und Darstellung von Frequenzgängen (Bode-Diagrammen oder Nyquist-Ortskurven) nutzen. Hinweise dazu entnehmen Sie bitte dem Abschnitt *Simulationen im Batch-Betrieb*.

Anzeige-fenster:



Im oberen Teil des Anzeigefensters sind die Verläufe von Eingangsgröße (Blockeingang 1, grün) und Ausgangsgröße (Blockeingang 2, blau) dargestellt. Die Statuszeile enthält die ermittelten Werte für Frequenz, Betrag und Phase. Der untere Teil des Fensters enthält bei mehreren aufeinanderfolgenden Simulationen (z. B. im Batch-Betrieb) das resultierende Bode-Diagramm bestehend aus Betragskennlinie (rot) und Phasenkennlinie (blaugrün). Über die Reset-Taste der Toolbar (Taste ganz links) lassen sich beide Anzeigen zurücksetzen (d. h. alle bisher aufgezeichneten Werte des Bode-Diagramms werden gelöscht).

Parameter-dialog:

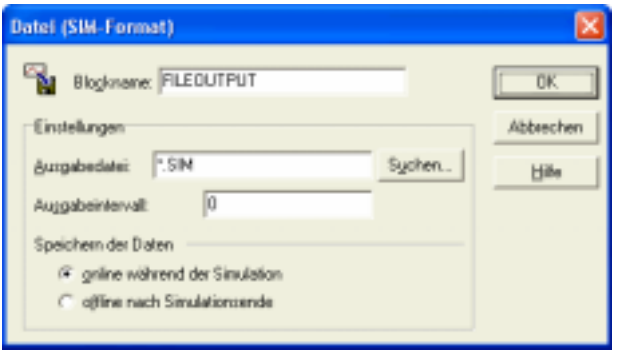
Ist die Option *Auto-Simulationsabbruch nach xxx Perioden* aktiviert, wird eine laufende Simulation automatisch abgebrochen, wenn die entsprechende Anzahl von Schwingungen erfasst wurde. Dadurch lässt sich insbesondere im Batch-Betrieb bei der Aufzeichnung von Frequenzgängen u. U. eine erhebliche Reduzierung der Gesamtrechenzeit erreichen. In einigen Fällen kann der vorzeitige Simulationsabbruch jedoch auch zu einer ungenauen Ermittlung der Phasenverschiebung führen; daher sollte diese Option mit Vorsicht benutzt werden.

Datei (File-Output)

Typname: FILEOUTPUT

Funktion: Ermöglicht die Ausgabe eines Signals $u(t)$ in Form von Wertepaaren (t_i, u_i) in eine Datei vom Typ SIM.

Parameter-dialog:



Wird für *Ausgabedatei* keine Extension angegeben, wird die Extension SIM benutzt. Über die Schaltfläche *Suchen* kann ein Dateieingabedialog angefordert werden. Der Parameter *Ausgabeintervall* gibt an, in welchem zeitlichen Abstand die einzelnen Wertepaare abgespeichert werden. Wird z. B. bei einer Simulationsschrittweite von 0.1 ein Ausgabeintervall von 0.5 gewählt, so wird lediglich jedes fünfte Wertepaar abgespeichert. Wird das Ausgabeintervall zu null gewählt (Voreinstellung), so wird jeder simulierte Wert auch abgespeichert. Das Speichern der Daten kann *online* während der Simulation oder *offline* nach Simulationsende erfolgen. Letztere Einstellung empfiehlt sich insbesondere bei Echtzeitsimulationen, da in diesem Fall während der Simulation kein zeitaufwendiger Zugriff auf die Festplatte erfolgen muss. Bei Offline-Speicherung werden die Daten im Arbeitsspeicher zwischengelagert.

Soll der Block im Rahmen von Batch-Läufen benutzt werden (siehe Abschnitt *Simulationen im Batch-Betrieb*), so ist der Einsatz der Steuersequenz *{#b}* innerhalb des Dateinamens sinnvoll, da ansonsten bei jeder Simulation die in der vorangegangenen Simulation des Batch-Laufes generierte Datei wieder überschrieben würde. Diese Steuersequenz wird im Batch-Betrieb durch die Nummer des aktuellen Simulationslaufes innerhalb des Batch-Laufes ersetzt.

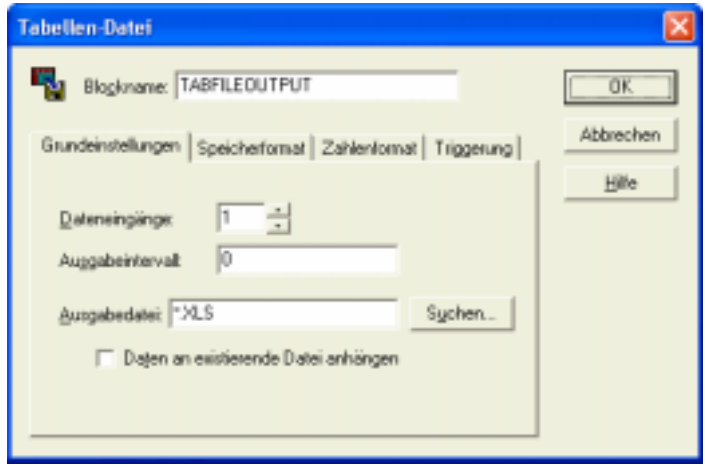
Beispiel: Als Name für die Ausgabedatei werde TEST{#b}.SIM vorgegeben. Wird nun ein Bach-Lauf mit insgesamt zehn Simulationen gestartet, werden nacheinander die Dateien TEST1.SIM, TEST2.SIM, ..., TEST10.SIM erzeugt. Im Kommentar jeder Datei stehen dabei automatisch Informationen über den Batch-Lauf und die benutzten Parameterwerte. Im Nicht-Batch-Betrieb wird die Steuersequenz einfach ignoriert; in diesem Fall würde also die Datei TEST.SIM erzeugt.

Tabellen-Ausgabedatei (Excel-Format)

Typname: TABFILEOUTPUT

Funktion: Dieser Block ermöglicht das Abspeichern von bis zu 49 Zeitverläufen in Tabellenform. Die abgespeicherten Dateien können dann auf einfache Weise direkt mit Tabellenkalkulationen wie z. B. Excel weiterverarbeitet werden. Die Dateien besitzen die Extension XLS. Die Abspeicherung der Daten kann periodisch oder extern getriggert erfolgen: Ist der Triggereingang C offen, erfolgt ein periodisches Abspeichern, ist er belegt, erfolgt ein Abspeichern wahlweise bei einer positiven Flanke am Triggereingang oder bei statischem High-Pegel.

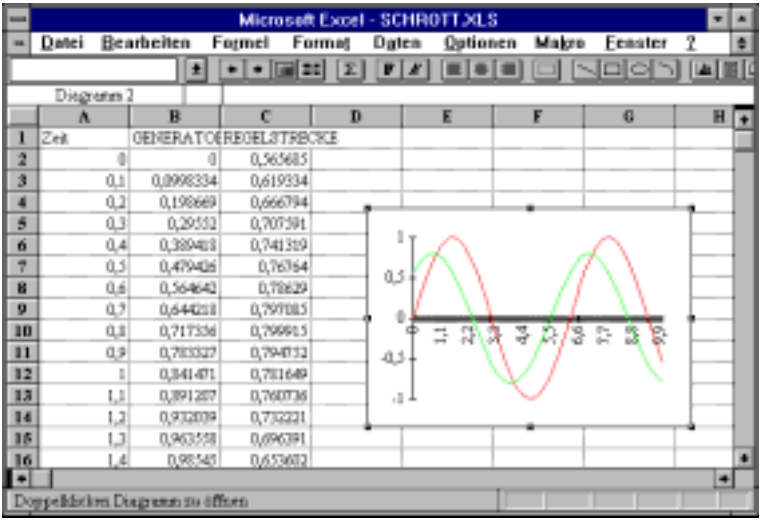
Parameterdialog:



Die Parameter haben folgende Bedeutung:

- *Dateneingänge* gibt die Anzahl der Blockeingänge (1 bis 49) an.
- *Ausgabedatei* gibt den Namen der Datei an, in der die Daten abgelegt werden. Über *Suchen* kann ein Dateieingabedialog angefordert werden. Die Zeitverläufe werden in dem Zeitabstand abgespeichert, der unter *Ausgabeintervall* angegeben ist. Beträgt dieser Wert 0, so werden die Werte bei jedem Simulationsschritt abgespeichert.
- Ist die Option *mit Spaltenüberschrift* aktiviert, so werden in der ersten Zeile der Datei die Bezeichner der den jeweiligen Eingangsgrößen entsprechenden Blöcke ausgegeben.

- Ist die Option *mit Zeitwerten* aktiviert, so enthält die erste Spalte der abgelegten Tabelle jeweils den aktuellen Zeitwert, andernfalls enthält sie die erste Zustandsgröße und eine Zeitparametrierung findet nicht statt.
- Über *Spaltentrennzeichen* wird angegeben, wie die einzelnen Spalten der Tabelle voneinander getrennt werden. Die Einstellungen unter *Dezimaltrennzeichen* geben an, welches Zeichen für die Darstellung des Dezimalpunkts gewählt wird.



Weiterverarbeitung von Dateien mit Excel

Soll der Block im Rahmen von Batch-Läufen benutzt werden (siehe Abschnitt *Simulationen im Batch-Betrieb*), so ist der Einsatz der Steuersequenz *{#b}* innerhalb des Dateinamens sinnvoll, da ansonsten bei jeder Simulation die in der vorangegangenen Simulation des Batch-Laufes generierte Datei wieder überschrieben würde. Diese Steuersequenz wird im Batch-Betrieb durch die Nummer des aktuellen Simulationslaufes innerhalb des Batch-Laufes ersetzt.

Beispiel: Als Name für die Ausgabedatei werde TEST{#b}.XLS vorgegeben. Wird nun ein Bach-Lauf mit insgesamt zehn Simulationen gestartet, werden nacheinander die Dateien TEST1.XLS, TEST2.XLS, ..., TEST10.XLS erzeugt. Im Kommentar jeder Datei stehen dabei automatisch Informationen über den Batch-Lauf und die benutzten Parameterwerte. Im Nicht-Batch-Betrieb wird die Steuersequenz einfach ignoriert; in diesem Fall würde also die Datei TEST.XLS erzeugt.

Block-Digitalanzeige

Typname: BLOCKDIGOUT

Funktion: Dieser Block realisiert eine Digitalanzeige mit frei wählbarem Ausgabeformat. Im Gegensatz zur normalen Digitalanzeige (Blocktyp DIGITALANZEIGE) erfolgt die Ausgabe bei diesem Blocktyp jedoch nicht in einem separaten Anzeigefenster, sondern direkt innerhalb des Blockes selbst.

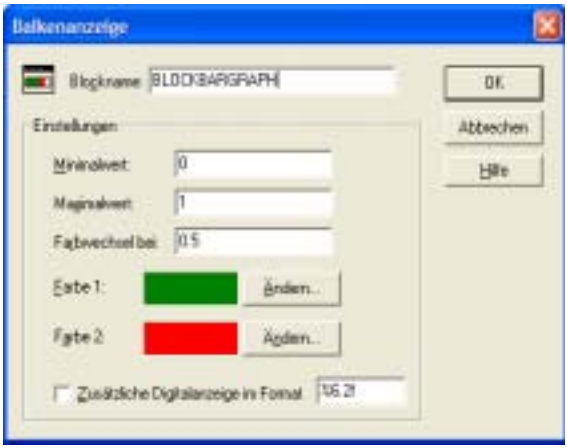
Parameter-dialog:

Block-Balkenanzeige

Typname: BLOCKBARGRAPH

Funktion: Dieser Block realisiert eine mehrfarbige Balkenanzeige mit zusätzlicher Digitalausgabe. Im Gegensatz zur normalen Balkenanzeige (Blocktyp BARGRAPH) erfolgt die Ausgabe bei diesem Blocktyp jedoch nicht in einem separaten Anzeigefenster, sondern direkt innerhalb des Blockes selbst.

Parameter-dialog:



Block-Statusanzeige

Typname: BLOCKSTATUS

Funktion: Dieser Block realisiert eine Statusanzeige. Im Gegensatz zur normalen Statusanzeige (Blocktyp STATUSANZEIGE) erfolgt die Ausgabe bei diesem Blocktyp jedoch nicht in einem separaten Anzeigefenster, sondern direkt innerhalb des Blockes selbst.

Parameter-dialog:

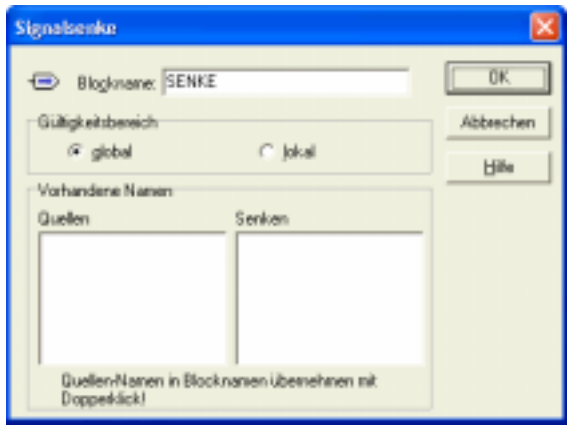
Signalsenke

Typname: SENKE

Funktion: Dieser Block dient zusammen mit dem Eingangsblocktyp *Signalquelle* zur Realisierung "drahtloser" Verbindungen zwischen Blöcken. Dabei "versendet" eine Signalsenke ihr Eingangssignal unter einem bestimmten Namen (dem Namen des Blocks). Dieses Signal kann dann an beliebigen - auch mehreren - Stellen in der Systemstruktur von einer Signalquelle mit gleichem Namen wieder empfangen werden. Groß- und Kleinschreibung der Blocknamen wird dabei nicht unterschieden. Signalsenken können *lokale* oder *globale* Gültigkeit haben. Während sie im ersten Fall nur innerhalb der zugehörigen System- bzw. Superblockdatei bekannt sind, gelten sie im globalen Fall auch über die Dateigrenzen hinaus. Der Einsatz beider Blocktypen empfiehlt sich insbesondere bei komplexen, stark vermaschten Systemstrukturen, um die Anzahl der sichtbaren Verbindungen gering zu halten.

Das Quellen/Senken-Konzept

Parameterdialog:



In der linken Listbox werden die Namen der bereits vorhandenen Quellen in alphabetischer Reihenfolge angezeigt, in der rechten Listbox die der Senken. Ein Quellen-Name kann durch Doppelklick direkt als Senken-Name übernommen werden.

Sonstige Systemblöcke

Parameter-Modifizierer

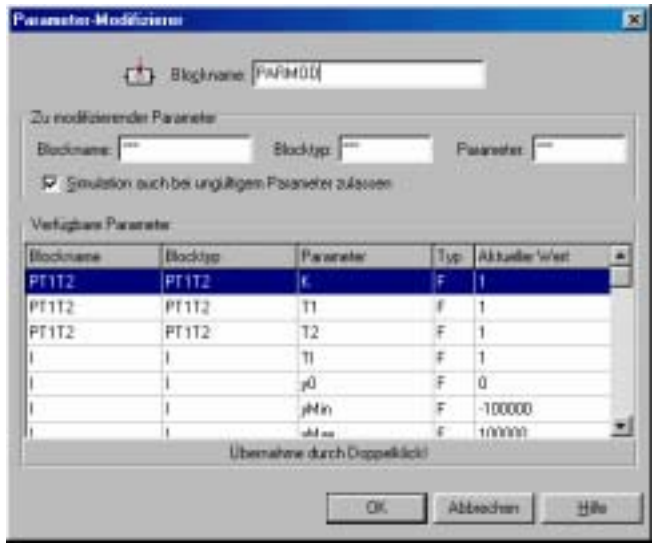
Typname: PARMOD

Funktion: Dieser Blocktyp erlaubt die signalgesteuerte Modifikation eines beliebigen (Fließkomma-) Exportparameters. Dadurch lassen sich aus der Simulation heraus gezielt Parameter von Blöcken (z. B. Verstärkungen oder Zeitkonstanten) ändern. Sofern am Freigabeeingang S des Blocks HIGH-Pegel anliegt (oder der Eingang offen gelassen wurde), wird der gewählte Exportparameter auf den am Dateneingang D anliegenden Wert gesetzt. Liegt der Freigabeeingang auf LOW-Pegel, findet keine Modifikation statt.

Um eine definierte Abarbeitungsreihenfolge zu erreichen, werden alle PARMOD-Blöcke jeweils am Ende des Simulationsschrittes abgearbeitet; der ggf. neue Parameterwert wird dann also erst zu Beginn des darauffolgenden Simulationsschrittes übernommen. Dies bedeutet insbesondere, dass im *ersten* Simulationsschritt grundsätzlich der im Parameterdialog des entsprechenden Blocks eingestellte Parameterwert benutzt wird.

Die während der Simulation über einen PARMOD-Block geänderten Parameter eines Blocks werden nach Beendigung der Simulation automatisch wieder auf ihre ursprünglichen Werte zurückgesetzt.

Parameter-dialog:



Ist die Option *Simulation auch bei ungültigem Parameter zulassen* aktiviert, so wird der Block auch dann abgearbeitet, wenn der gewählte Parameter (z. B. aufgrund einer zwischenzeitlich geänderten Bezeichnung des entsprechenden Blocks) ungültig ist (der Block ist dann ohne Funktion). Ist die Option nicht aktiviert, erfolgt eine Fehlermeldung und die Simulation kann nicht durchgeführt werden.

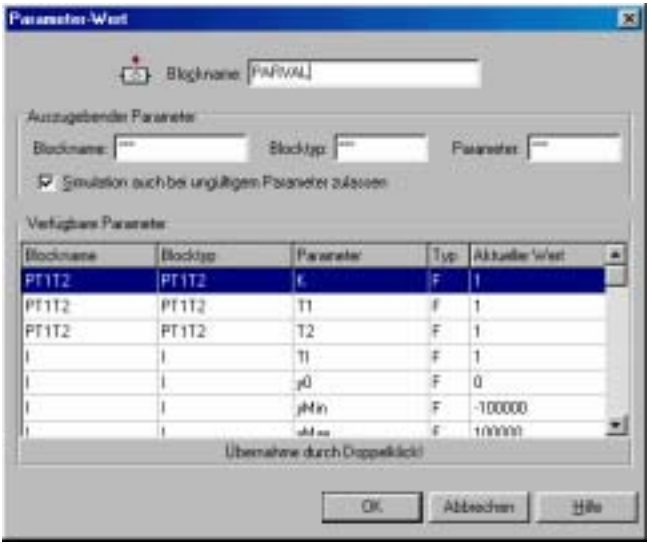
Hinweis: Eine Überprüfung des Parameterwertes auf Zulässigkeit (z. B. Bereichsgrenzenüberschreitung) vor der Übertragung in den entsprechenden Zielblock erfolgt *nicht*; sie muss also ggf. durch eine Modifikation der Simulationsstruktur (z. B. Vorschalten eines Begrenzers vor den Dateneingang des PARMOD-Blocks) realisiert werden.

Parameter-Wert

Typname: PARVAL

Funktion: Dieser Blocktyp gibt an seinem Ausgang den aktuellen Wert eines beliebigen (Fließkomma-) Exportparameters aus.

Parameter-dialog:



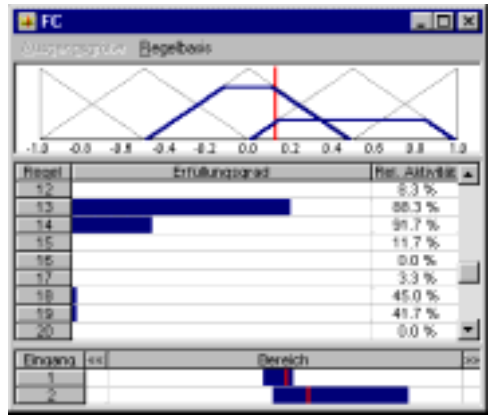
Ist die Option *Simulation auch bei ungültigem Parameter zulassen* aktiviert, so wird der Block auch dann abgearbeitet, wenn der gewählte Parameter (z. B. aufgrund einer zwischenzeitlich geänderten Bezeichnung des entsprechenden Blocks) ungültig ist (der Block ist dann ohne Funktion). Ist die Option nicht aktiviert, erfolgt eine Fehlermeldung und die Simulation kann nicht durchgeführt werden.

Fuzzy Controller

Typname: FC

Funktion: Dieser Systemblock realisiert einen Fuzzy Controller, der über eine FUZ-Datei parametriert wird, die mit Hilfe der WinFACT-Fuzzy-Shell FLOP erstellt werden kann. Der Fuzzy Controller-Block besitzt ein als *Fuzzy-Debugger* bezeichnetes, abschaltbares Kontrollfenster, das während der Simulation aus seiner Symboldarstellung geholt werden kann und wichtige Hinweise auf die innere Funktion des Controllers gibt. Der Fuzzy-Debugger bietet damit eine wesentliche Unterstützung beim interaktiven Fuzzy Controller-Entwurf. Die nachfolgenden Grafiken zeigen den Aufbau des Debuggers sowie seinen Parameterdialog. Der Fuzzy-Debugger zeigt online während der Simulation folgende Größen an:

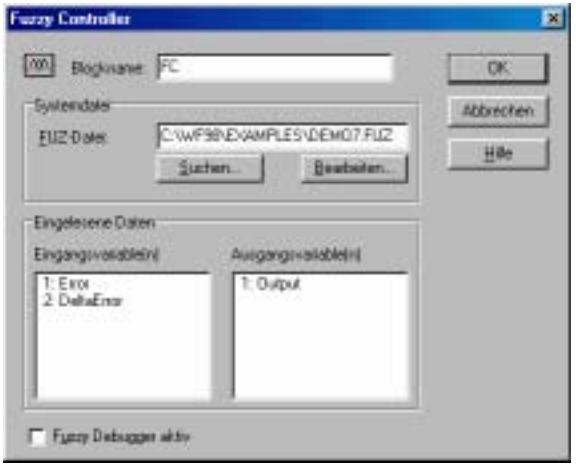
- Im oberen Fensterdrittel die Fuzzy-Mengen der gewählten FC-Ausgangsgröße (grau), die aktuelle scharfe Ausgangsgröße (rot) sowie die resultierende Ausgangs-Fuzzy-Menge (blau).
- Im mittleren Fensterdrittel die aktuellen Erfüllungsgrade aller Regeln (blaue Balken) sowie ihre relative Aktivität. Besitzt eine Regel z. B. eine relative Regelaktivität von 30%, so bedeutet dies, dass die Regel in 30% aller Simulationsschritte einen Erfüllungsgrad größer null aufgewiesen hat.
- Im unteren Fensterdrittel die aktuellen scharfen Eingangswerte (rot) sowie die bisherige Ausnutzung des Eingangsgrößenbereichs des Fuzzy Controllers (blau). Bei Unter- bzw. Überschreitung des jeweiligen Bereichs wechseln die Anzeigen links bzw. rechts der Bereichsanzeige ihre Farbe auf gelb.



Der Fuzzy-Debugger

Über die Menüoption AUSGANGSGRÖßE kann die darzustellende Ausgangsgröße des Fuzzy Controllers gewählt werden. Die Menüoption REGELBASIS ermöglicht zu Kontrollzwecken die Ausgabe der zugrundeliegenden Regelbasis.

Parameter-dialog:



Wird für *Systemdatei* keine Extension angegeben, wird die Extension FUZ benutzt. Über die Schaltfläche *Suchen* kann ein Dateieingabedialog angefordert werden. Die zugehörigen Ein- und Ausgangsvariablen werden danach zur Kontrolle in den Listenfenstern angezeigt. Über den Schalter *Bearbeiten* kann die angegebene Datei direkt zum Bearbeiten geöffnet werden. Nach dem Verlassen des Dialogs wird die Anzahl der Blockein- und -ausgänge - sofern erforderlich - automatisch angepasst. Der Fuzzy Debugger kann bei Bedarf zur Erzielung einer höheren Simulationsgeschwindigkeit deaktiviert werden.

Online-Fuzzy Controller

Typname: FCONLINE

Funktion: Dieser Systemblock realisiert einen mit der WinFACT-Fuzzy-Shell FLOP entworfenen Fuzzy Controller, der online über DDE mit BORIS kommuniziert. Einzelheiten dazu entnehmen Sie bitte Kapitel 7 der Dokumentation, *Entwurf von Fuzzy-Systemen mit der Fuzzy Shell FLOP*.

NeuroModel-Block

Typname: NEUROMODEL

Funktion: Dieser Blocktyp realisiert ein mit NeuroModel® generiertes Neuronales Netz. Einzelheiten entnehmen Sie bitte der Dokumentation zu NeuroModel.

Parameter-dialog:

*fuzzy*TECH-Block

Typname: FTRUN

Funktion: Dieser Blocktyp realisiert einen mit *fuzzy*TECH® generierten Fuzzy Controller. Einzelheiten entnehmen Sie bitte der Dokumentation zu *fuzzy*TECH.

Parameter-dialog:

ALASKA 4-Modell

Typname: ALASKA

Funktion: Dieser Blocktyp ermöglicht das Einbinden eines mit dem Mehrkörper-Simulationssystem *ALASKA 4* erstellten Modells. Einzelheiten entnehmen Sie bitte der Dokumentation zu *ALASKA*.

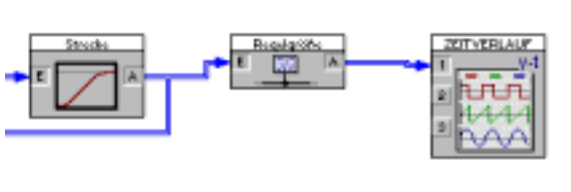
Parameter-dialog:

Label

Typname: LABEL

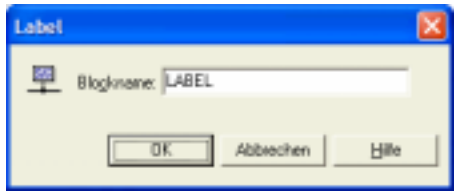
Funktion: Blöcke vom Typ *Label* haben keinerlei technische Funktion, sondern dienen lediglich dazu, Signalen einen bestimmten Namen zu geben, unter dem sie dann beispielsweise in Zeitverlaufsblöcken oder - besonders wichtig - in Superblöcken auftreten (siehe Abschnitt *Arbeiten mit Superblöcken*).

Beispiel: Die Ausgangsgröße eines PT_1-Gliedes mit Namen *Strecke* soll auf einen Zeitverlaufsblock gegeben, dort aber als *Regelgröße* bezeichnet werden. Um dies zu erreichen, setzt man zwischen PT_1-Glied und Zeitverlauf ein Label mit Namen *Regelgröße* (siehe nachfolgende Grafik).



Einsatz von Label-Blöcken

Parameter-dialog:

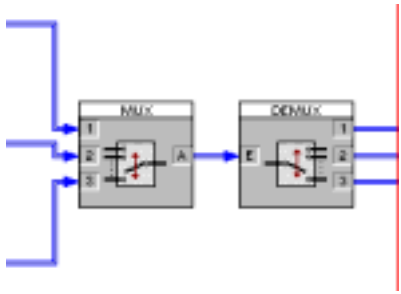


Multiplexer

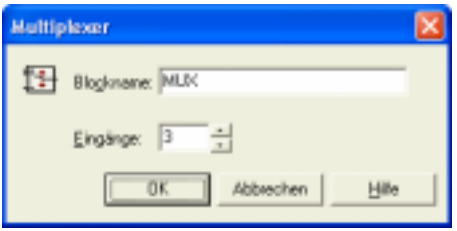
Typname: MUX

Funktion: Dieser Block ermöglicht die Zusammenfassung mehrerer Verbindungen zu einer einzigen. Sein Einsatz ist daher dann sinnvoll, wenn mehrere Verbindungen gleichzeitig über einen größeren Bereich gezogen werden können. Die Ausgangsverbindung eines Multiplexers muss mit dem Eingang eines *Demultiplexer*-Blocks verbunden werden.

Beispiel:



Parameter-dialog:



Demultiplexer

Typname: DEMUX

Funktion: Dieser Block splittet die von einem *Multiplexer*-Block erzeugte Mehrfachverbindung wieder auf. Die Eingangsverbindung eines Demultiplexers muss daher vom Ausgang eines Multiplexers stammen.

Parameter-dialog:

Dokumentanzeige

Typname: VIEWDOC

Funktion: Dieser Block ermöglicht die Anzeige eines Dokumentes (z. B. einer PDF-Datei) mit Hilfe des in der Windows-Registry für den entsprechenden Dateityp festgelegten Programms. Die Anzeige erfolgt bei Betätigung der im Block befindlichen Schaltfläche.

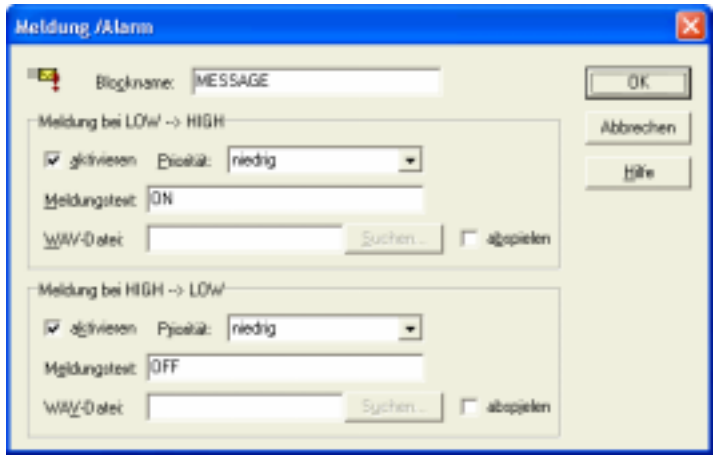
Parameter-dialog:

Meldung/Alarm

Typname: MESSAGE

Funktion: Dieser Block ermöglicht die Erzeugung einer Meldung bzw. eines Alarms, sofern an seinem Eingang eine positive oder negative Flanke auftritt. Jede Meldung kann neben der grafischen Anzeige durch das Abspielen einer WAV-Datei gekennzeichnet werden. Einzelheiten dazu siehe im Abschnitt *Verwaltung von Meldungen und Alarmen*.

Parameter-dialog:

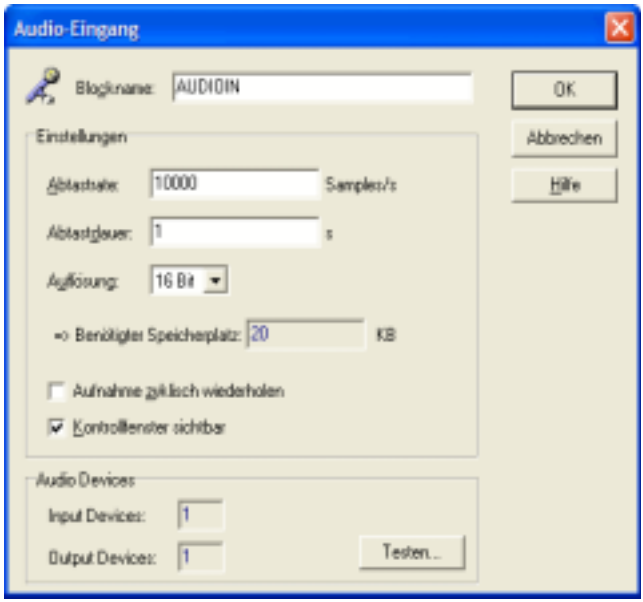


Audio-Eingang

Typname: AUDIOIN

Funktion: Dieser Block ermöglicht das Einlesen von Audiosignalen über den Mikrofoneingang der Soundkarte. Das eingelesene Signal wird auf Wunsch in einem separaten Kontrollfenster visualisiert. Die Beispieldatei *AudioIn.bsy* erläutert die Anwendung dieses Systemblocks.

Parameterdialog:



Audio-Ausgang

Typname: AUDIOOUT

Funktion: Dieser Block ermöglicht die Ausgabe von Audiosignalen über die Soundkarte. Das ausgegebene Signal wird auf Wunsch in einem separaten Kontrollfenster visualisiert. Die Beispieldatei *AudioOut.bsy* erläutert die Anwendung dieses Systemblocks.

Parameter-dialog:

Hardware-Interface

Typname: HARDWARE

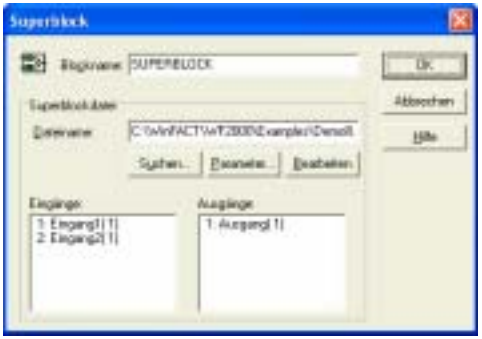
Funktion: Dieser Block dient als Interface zu Hardware jeglicher Art (z. B. PC-Einsteckkarten, externe Hardware am RS-232-Port etc.) und kann sowohl Hardware-Eingänge als auch Hardware-Ausgänge enthalten. Hinweise dazu entnehmen Sie bitte der Dokumentation zum jeweiligen Hardware-Treiber.

Superblock

Typname: SUPERBLOCK

Funktion: Ein Superblock enthält ein beliebiges Teilsystem aus Blöcken und Verbindungen. Der Arbeit mit Superblöcken ist ein eigener Abschnitt *Arbeiten mit Superblöcken* gewidmet.

Parameterdialog:



Wird für *Dateiname* keine Extension angegeben, wird die Extension SBL benutzt. Über die Schaltfläche *Suchen* kann ein Dateieingabedialog angefordert werden. Die zugehörigen Ein- und Ausgangsvariablen werden danach zur Kontrolle in den Listenfenstern angezeigt. Die Schaltfläche *Parameter...* ermöglicht die Bearbeitung der Superblock-Exportparameter. Über den Schalter *Bearbeiten* kann die angegebene Datei direkt zum Bearbeiten geöffnet werden. Nach dem Verlassen des Dialogs wird die Anzahl der Blockein- und -ausgänge - sofern erforderlich - automatisch angepasst.

Benutzerdefinierter Block (User-DLL)

Typname: USER1

Funktion: Benutzerdefinierter Block im Windows-DLL-Format. Einzelheiten dazu finden Sie im Abschnitt *Benutzerdefinierte Systemblöcke*.

Parameter-dialog:

Wird für *Systemdatei* keine Extension angegeben, wird die Extension DLL benutzt. Über die Schaltfläche *Suchen* kann ein Dateieingabedialog angefordert werden. Die Schaltfläche *Öffnen* liest die angegebene Datei ein. Die zugehörigen Ein- und Ausgangsvariablen werden danach zur Kontrolle in den Listenfenstern angezeigt. Nach dem Verlassen des Dialogs wird die Anzahl der Blokkein- und -ausgänge - sofern erforderlich - automatisch angepasst. Über die Schaltfläche *Dialog...* wird der blockspezifische Parameterdialog aufgerufen.

Arbeiten mit Superblöcken

Was ist ein Superblock?

Ein Superblock ist ein spezieller Systemblocktyp, der durch *Gruppierung mehrerer Systemblöcke und ihrer Verbindungen* entsteht. Der Superblock stellt also nichts anderes als ein Teilsystem dar, das zu einem neuen Block - in der Regel mit Ein- und Ausgängen - zusammengefasst wird. Von außen betrachtet hat der Superblock dann eine Art "Black-Box"-Charakteristik. Superblöcke eignen sich damit insbesondere für die übersichtliche Strukturierung komplexer Systeme sowie zur Gruppierung häufig benutzter Teilsysteme.

Beispiel: Sie möchten für einen zuvor modellierten Prozess - der beispielsweise aus der Reihenschaltung dreier Systemblöcke besteht - verschiedene Regler testen. Dann wandeln Sie zunächst die Reihenschaltung in einen Superblock um, auf den Sie dann später aus beliebigen Systemen zugreifen können.

Wie werden Informationen über den internen Aufbau des Superblocks abgelegt?

Superblöcke in BORIS sind *dateireferenziert*: Alle Informationen über die im Superblock enthaltenen Blöcke und Verbindungen werden in einer Datei mit der Extension SBL abgelegt. Diese Dateien sind - bis auf einen in Superblockdateien zusätzlich enthaltenen Dateiheader - mit "normalen" BORIS-Systemdateien vom Typ BSY identisch. Somit lassen sich Superblockdateien nach dem Laden auch als gewöhnliche Systemdateien speichern und umgekehrt, ebenso können sie natürlich eigenständig simuliert werden.

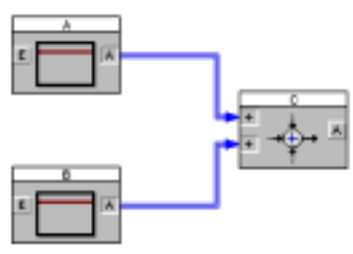
Der Vorteil der Dateireferenzierung liegt auf der Hand: Wird in einer Superblockdatei eine Änderung vorgenommen, so sind automatisch alle Systemdateien, die diesen Superblock einbinden, aktualisiert. Der Superblock selbst ist über die Angabe seines Dateinamens eindeutig definiert.

Ein- und Ausgänge von Superblöcken

Die Ein- und Ausgänge eines Superblocks werden von BORIS automatisch festgelegt. Dabei gilt es folgendes zu beachten:

- Alle offenen Systemblockein- bzw. -ausgänge werden zu Ein- bzw. Ausgängen des Superblocks.

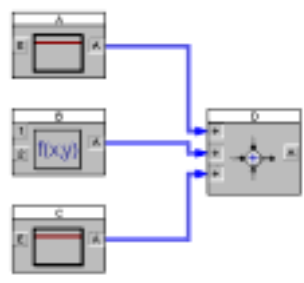
  Beispiel: Nachfolgende Struktur soll zu einem Superblock gruppiert werden.



Der resultierende Superblock besitzt zwei Eingänge (die der Blöcke A und B) sowie einen Ausgang (den von Block C).

- Sollen Blockausgänge, die bereits eine Verbindung enthalten (zum Beispiel eine Rückführung), zu Superblockausgängen werden, so setzt man zweckmäßigerweise Label-Blöcke ein (siehe Abschnitt *Superblöcke und Labels*). Sollen im umgekehrten Fall offene Systemeingänge nicht zu Superblockeingängen werden, so beschalten Sie diese einfach z. B. mit einem Konstanten-Block, der auf null (oder einen anderen geeigneten Wert) gesetzt wird.

- Die Ein- bzw. Ausgänge des Superblocks werden in der Reihenfolge durchnummeriert, in der die entsprechenden Blöcke eingefügt wurden.
  Beispiel:



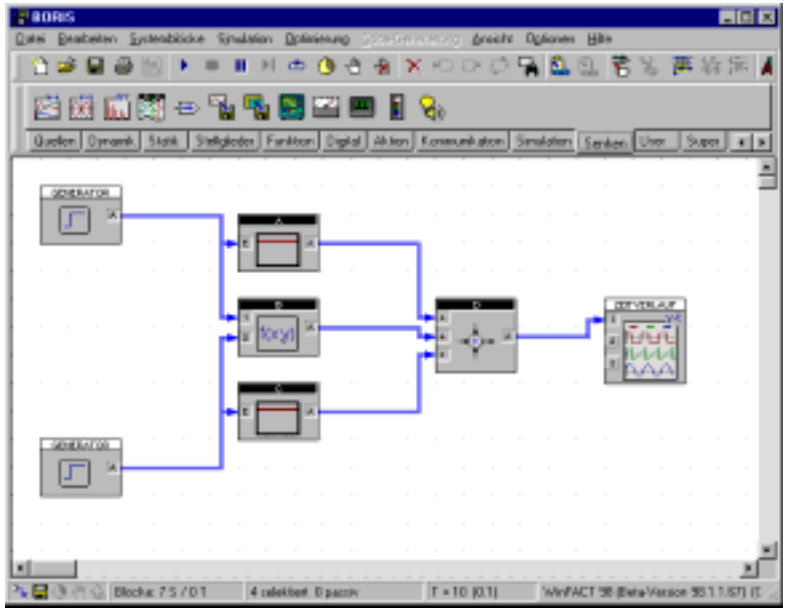
Wurden die Blöcke in der Reihenfolge A, B, C, D eingefügt, so wird der Eingang von Block A zum ersten, der erste Eingang von Block B zum zweiten, der zweite Eingang von Block B zum dritten und der Eingang von Block C zum vierten Superblockeingang. Analoges gilt bei Vorliegen mehrerer Ausgänge. Sollen Superblockein- und -ausgänge vertauscht werden, kann man entweder die entsprechenden Systemblöcke zunächst löschen und dann wieder in geeigneter Reihenfolge einfügen oder aber *Labels* einfügen (siehe Abschnitt *Superblöcke und Labels*).

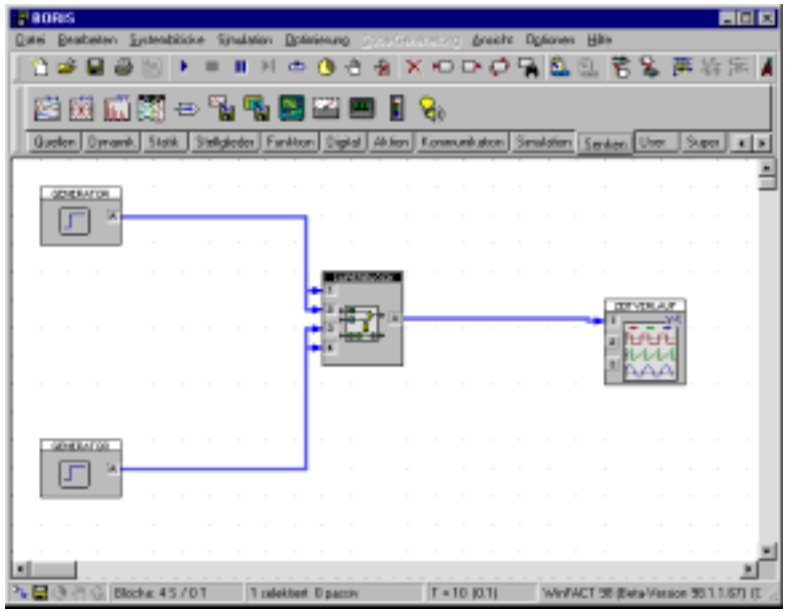
Um einen neuen Superblock zu definieren...

... haben Sie grundsätzlich zwei Möglichkeiten:

- Sie erstellen den Superblock zunächst als separate Datei, speichern ihn über DATEI | DATEI SPEICHERN UNTER... ab und laden ihn dann später über die Angabe des Dateinamens in das entsprechende übergeordnete System.
- Soll ein Teilsystem einer bereits konfigurierten Systemstruktur in einen Superblock überführt werden, so selektieren Sie zunächst die zu gruppierenden Blöcke und nehmen dann die eigentliche Gruppierung über BEARBEITEN | GRUPPIEREN ZU SUPERBLOCK oder die Schaltfläche  der System-Toolbar vor. BORIS fragt Sie dann nach dem Namen, den die Superblockdatei haben soll und fügt anschließend den Superblock ein (siehe nachfolgende Grafiken).

Ein Superblock lässt sich durch Selektieren und BEARBEITEN | SUPERBLOCK AUFLÖSEN oder die Schaltfläche  jederzeit wieder in seine Bestandteile zerlegen. Dabei ist darauf zu achten, dass im "Umkreis" des Superblocks genügend Platz zur Verfügung steht.



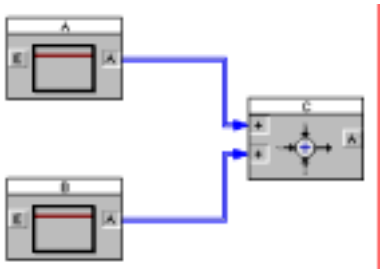


Gruppieren von Blöcken zu einem Superblock: Selektieren der Blöcke (oben) und Bildschirm nach Umwandlung in einen Superblock (unten)

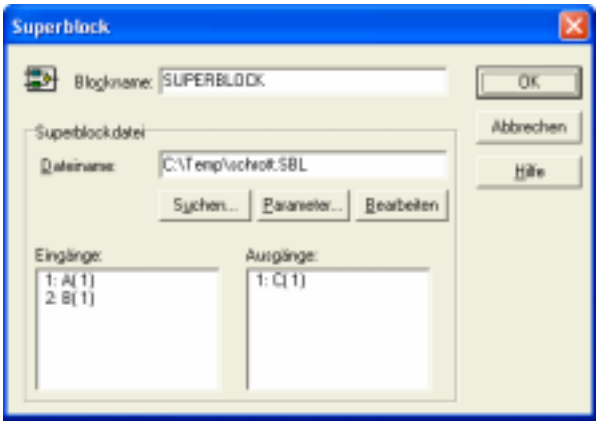
Superblöcke und Labels

Der Einsatz von Label-Blöcken kann im Zusammenhang mit Superblöcken an verschiedenen Stellen sinnvoll sein:

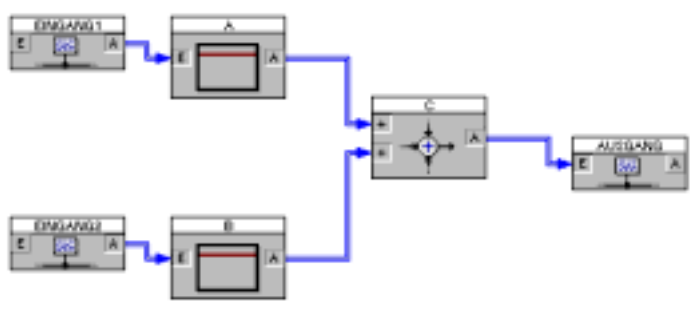
- Zur Umbenennung der Superblockein- bzw. -ausgänge. Damit der Anwender weiß, welcher Ein- bzw. Ausgang des Superblocks welche Funktion hat, auch ohne dass er die Superblockdatei zunächst zur Betrachtung laden muss, erhalten alle Superblockein- und -ausgänge einen Namen, der im Parameterdialog des Superblocks angezeigt wird. Als Voreinstellung wird der Name des intern mit dem Ein- bzw. Ausgang verbundenen Blocks in Kombination mit dessen Ein- bzw. Ausgangsnummer benutzt. Betrachten Sie dazu folgendes Beispiel:



Wird dieses Teilsystem in einen Superblock verwandelt, so werden die Eingänge des Superblocks mit *A(1)* bzw. *B(1)* bezeichnet, der Ausgang mit *C(1)*. Der Parameterdialog des Superblocks sieht dann also wie folgt aus:



Um den Ein- und Ausgängen griffigere Namen zu geben, kann man ihnen nun Labels vor- bzw. nachschalten, die dann mit den gewünschten Namen bezeichnet werden. Die modifizierte Superblockstruktur sieht dann wie folgt aus:



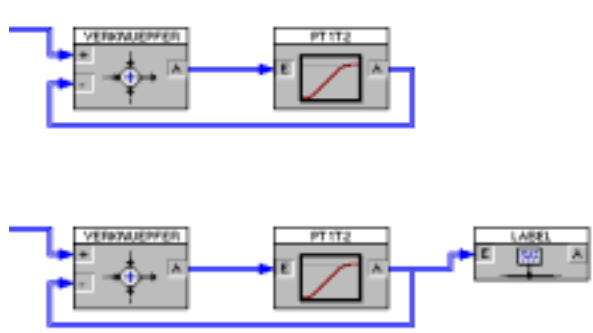
Die Eingänge wurden hier mit *Eingang1* bzw. *Eingang2* bezeichnet, der Ausgang mit *Ausgang* (in der Praxis sollte man möglichst sinnvollere Namen wählen!). Ruft man nunmehr den Parameterdialog des Superblocks auf, erscheinen dort die entsprechenden Einträge.

Ändern der Reihenfolge

- Für die Umnummerierung von Ein- oder Ausgängen. Sollen die Superblockein- oder -ausgänge eine neue Reihenfolge erhalten, fügt man einfach entsprechende Labels in der gewünschten Reihenfolge ein.

Erzeugung offener Ausgänge

- Zur Erzeugung offener Ausgänge. Soll ein Ausgang der Superblockstruktur, der bereits eine Verbindung enthält, zu einem Superblockausgang werden, so setzen Sie hinter den Ausgang ein Label und erzeugen so einen offenen Ausgang. Nachfolgende Grafik zeigt ein derartiges Beispiel.



Um den Ausgang des PT1T2-Glieds (oben) zu einem Superblockausgang zu machen, wird ihm ein Label nachgeschaltet (unten).

Ausgangsblöcke in Superblöcken

Alle Ausgangsblöcke (z. B. Zeitverlauf, Oszillograph usw.) können ohne weiteres auch innerhalb von Superblöcken verwendet werden. Dabei ist jedoch zu beachten, dass das entsprechende Anzeigefenster *nur während der Simulation* sichtbar ist! Es erscheint also bei Start der Simulation und verschwindet unmittelbar nach Beendigung der Simulation. Eine "Nachbetrachtung" der Simulationsergebnisse ist in diesen Fällen also nicht möglich. Das gleiche gilt auch für eventuelle Anzeigefenster anderer Blöcke (z. B. den Fuzzy-Debugger des Fuzzy Controllers).

Quellen und Senken in Superblöcken

Auch Signalquellen und -senken können in Superblöcken wie gewohnt verwendet werden. In der Regel wird man beide Blocktypen dabei in der Betriebsart *lokal* betreiben, so dass Ihre Gültigkeit auf den jeweiligen Superblock beschränkt bleibt. Prinzipiell können aber auch *globale* Quellen und Senken in einem Superblock benutzt werden. Dies bedeutet einerseits, dass ein Signal, das von einem Superblock über eine *globale* Signalsenke versendet wird, auch außerhalb des Superblocks mit Hilfe einer zugehörigen *globalen* Signalquelle empfangen werden kann. Andererseits können Superblöcke über interne, *globale* Signalquellen alle Signale empfangen, die von *globalen* Signalsenken außerhalb des Blocks generiert werden.

Vorsicht: Wird derselbe Superblock mehrmals in einer Struktur eingesetzt, sollte auf *globale* Signalsenken innerhalb des Superblocks in jedem Fall verzichtet werden, da ansonsten mehrere Senken mit demselben Namen existieren! Dies führt bei der Simulation in der Regel zu unerwünschten "Effekten"!

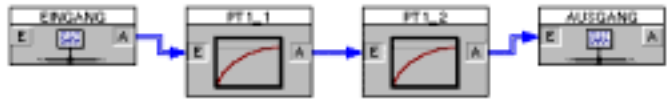
Superblöcke in Superblöcken in Superblöcken...

Ein System kann beliebig viele Superblöcke enthalten. Ein Superblock selbst kann ebenfalls einen oder mehrere Superblöcke enthalten usw. - die Verschachtelungstiefe ist beliebig. Nicht erlaubt sind selbstverständlich rekursive Referenzen von Superblöcken: Enthält Superblock A den Superblock B, so darf letzterer nicht gleichzeitig wieder auf Block A zugreifen.

Exportieren von Parametern

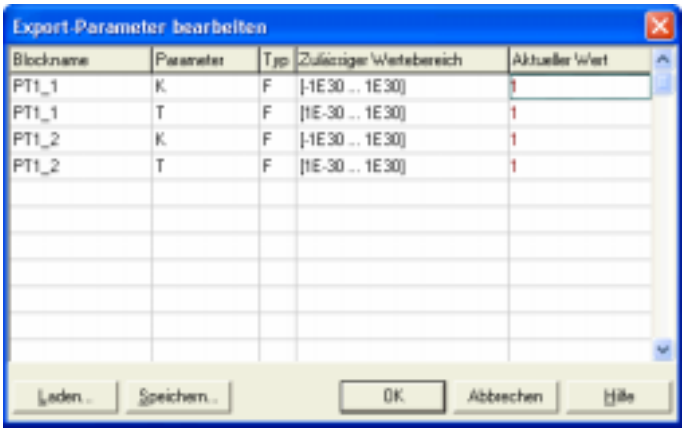
Häufig möchte man zwar dieselbe Struktur mehrfach verwenden, jedoch jedes Mal mit unterschiedlichen Parametern. BORIS bietet für solche Zwecke die Möglichkeit, die Exportparameter von Blöcken "nach außen" zu führen, so dass sie von der nächsthöheren Systemebene modifiziert werden können. Dazu sind lediglich innerhalb der betreffenden Superblockdatei die entsprechenden Exportparameter zu aktivieren; sie können dann - unabhängig von ihrem Wert innerhalb der Superblockdatei – bei jeder Nutzung des Superblocks unabhängig gesetzt werden.

Beispiel: Ein einfacher Superblock bestehe aus der Reihenschaltung zweier PT1-Glieder (siehe nachfolgende Grafik). Dieser Superblock soll nun von anderen Strukturen benutzt werden, wobei Verstärkungen und Zeitkonstanten jeweils innerhalb der aufrufenden Struktur modifiziert werden sollen. Dazu müssen lediglich in den Parameterdialogen der beiden PT1-Glieder jeweils die Exportparameter aktiviert werden (siehe auch Beispieldatei SUPEREXP.BSY im Examples-Verzeichnis).



Superblock mit Exportparametern

Nach dem Speichern lässt sich der Superblock wie gewohnt in eine beliebige Systemstruktur einbinden, wobei nunmehr aber jeweils die exportierten Parameter *für jeden Superblock getrennt* modifiziert werden können. Dazu dient die Schaltfläche *Parameter...* innerhalb des Superblock-Parameterdialogs. Die hier vorgegebenen Parameter werden dann nicht innerhalb der Superblockdatei, sondern innerhalb der aufrufenden Datei gespeichert. Da die Zuordnung der Exportparameter anhand des Blocknamens vorgenommen wird, ist es wichtig, alle exportierenden Blöcke mit eindeutigen Namen zu versehen. Dies kann vorab über DATEI | AUF DOPPELTE BLOCKNAMEN ÜBERPRÜFEN... überprüft werden.

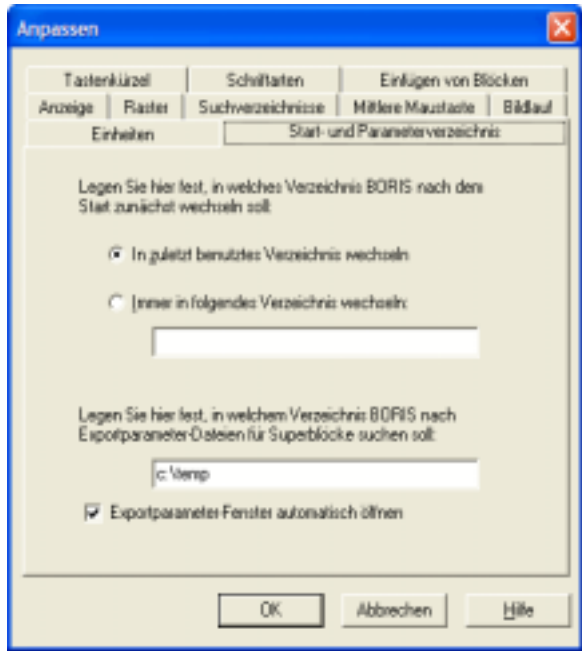


Zugehöriger Dialog zur Modifikation der Superblock-Exportparameter

Beachten Sie bitte, dass der Export von Superblock-Parametern *nur in die nächsthöhere* Ebene, nicht aber darüber hinaus möglich ist. Wird z. B. ein Superblock A von einem Superblock B benutzt, so sind dort die Exportparameter des Superblocks A modifizierbar. Wird jedoch Superblock B seinerseits in eine Struktur eingebunden, so kann von dort aus auf die Exportparameter von Superblock A nicht mehr zugegriffen werden!

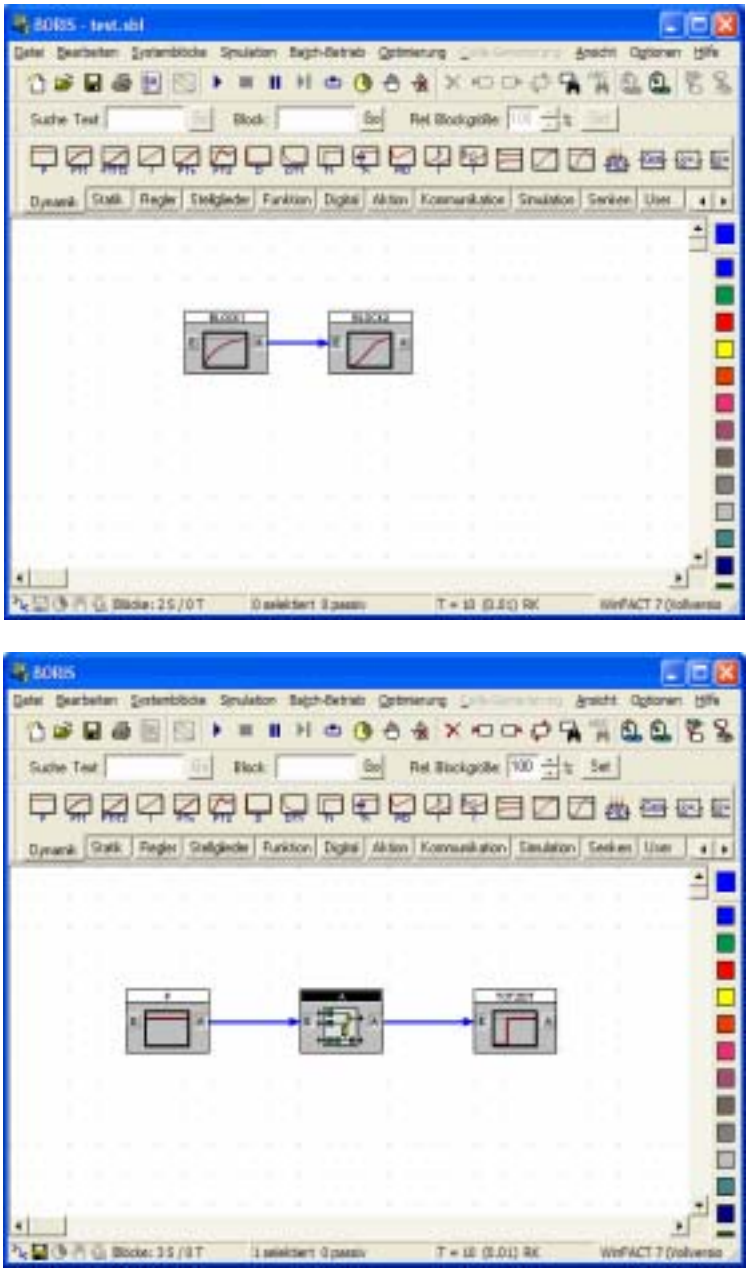
Einlesen von Exportparametern aus Datei

Alternativ zur manuellen Eingabe über Dialog können die Exportparameter eines Superblocks auch aus einer Textdatei gelesen werden. Diese Textdatei muss den selben Namen besitzen wie der zugrundeliegende Superblock und die Extension EXP. Sie muss sich in demjenigen Verzeichnis befinden, dass unter OPTIONEN | ANPASSEN... auf der Palette *Start- und Parameterverzeichnis* festgelegt wurde (siehe nachfolgende Bildschirmgrafik).



Festlegung des Verzeichnisses für Exportparameter-Dateien (hier c:\temp*)*

Der Aufbau der Textdatei soll an einem Beispiel erläutert werden. Nachfolgende Bildschirmgrafiken zeigen einen Superblock bestehend aus der Reihenschaltung eines PT1- und eines PT1T2-Gliedes, bei denen die Parameter jeweils als Exportparameter freigeschaltet wurden (obere Grafik). Die beiden Blöcke tragen hier die Namen *BLOCK1* und *BLOCK2*. Die untere Grafik zeigt nun eine Systemstruktur, in der dieser Superblocks (Mitte) eingebunden ist; der zugehörige Block trägt hier den Blocknamen *A*.



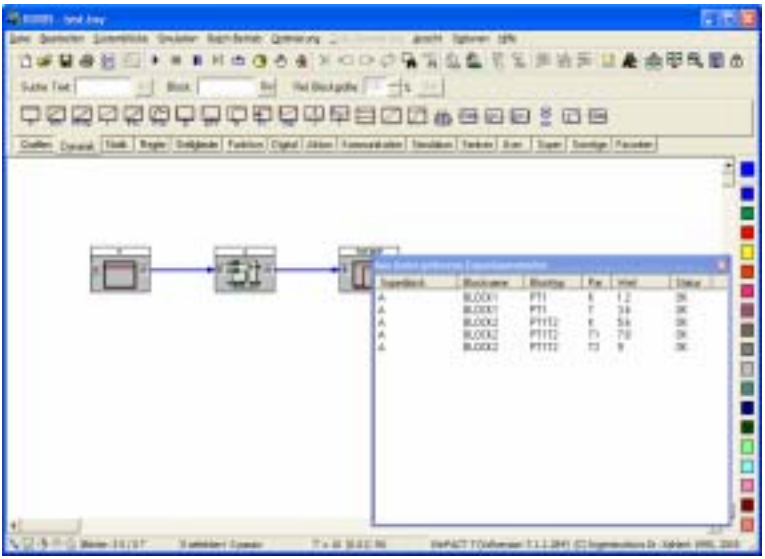
Interner Aufbau des Superblocks (oben) und Struktur mit eingebundenem Superblock (unten)

Nachfolgendes Listing zeigt den Aufbau der zugehörigen Exportparameterdatei, die den Namen A.EXP tragen muss. Diese Datei enthält für im Superblock enthaltenen Block, der Parameter exportiert, eine eigene *Sektion*, die jeweils durch einen in eckige Klammern eingeschlossenen Bezeichner eingeleitet wird. Dieser Bezeichner setzt sich zusammen aus dem jeweiligen Blocknamen, einem senkrechten Strich und dem jeweiligen Blocktyp. Danach folgen zeilenweise die einzulesenden Exportparameter, wobei jeweils der Name des Parameters gefolgt von einem Gleichheitszeichen und dem zu setzenden Wert angegeben werden muss. In untenstehendem Beispiel wird also die Verstärkung *K* vom *PT1*-Block *BLOCK1* auf 1.2 und seine Zeitkonstante *T* auf 3.4 gesetzt; die Verstärkung *K* des *PT1T2*-Blocks *BLOCK2* wird auf 5.6, seine erste Zeitkonstante *T1* auf 7.8 und seine zweite Zeitkonstante *T2* auf 9 gesetzt.

```
[BLOCK1|PT1]
K=1.2
T=3.4

[BLOCK2|PT1T2]
K=5.6
T1=7.8
T2=9
```

Beim Starten der Simulation werden die aus der Datei gelesenen Exportparameter in einem separaten Fenster protokolliert, sofern diese Option nicht unter OPTIONEN | ANPASSEN... deaktiviert wurde (siehe oben). Nachfolgende Bildschirmgrafik zeigt die Anzeige für das vorliegende Beispiel. Konnte ein Parameter *nicht* eingelesen werden (weil er versehentlich oder auch bewusst nicht in die Datei aufgenommen wurde), so wird sein Status im Protokollfenster mit *Failed* statt *OK* angegeben. Es wird dann für diesen Parameter der manuell eingegebene Wert benutzt.

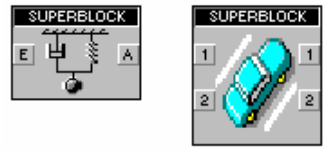


Anzeige der eingelesenen Exportparameter beim Starten der Simulation

Benutzerdefinierte Block-Bitmaps

Optional hat der Anwender die Möglichkeit, jeden seiner Superblöcke mit einem *blockspezifischen Bitmap* zu versehen. Dieses Block-Bitmap wird als BMP-Datei erstellt und muss denselben Namen aufweisen wie die Superblockdatei selbst, jedoch mit der Extension BMP, und sich im selben Verzeichnis befinden wie der Superblock. Zu einem Superblock mit dem Namen MOTOR.SBL gehört also die Bitmap-Datei MOTOR.BMP. BORIS überprüft beim Einfügen eines Superblocks automatisch, ob die zugehörige BMP-Datei vorhanden ist. Ist dies der Fall, wird diese benutzt, ansonsten das Standard-Superblock-Bitmap. Das Bitmap sollte eine Größe von 48×42 Pixeln aufweisen; Bitmaps anderer Größe werden automatisch gedehnt bzw. gestaucht, so dass sie optimal in das Blockschaltbild passen. Auch für die Druckerausgabe kann ein benutzerdefiniertes Bitmap (möglichst als s/w-Bitmap) definiert werden. Dieses muss am Ende des Dateinamens zusätzlich die Kennung *_P* erhalten.

Beispiele: zu MOTOR.SBL gehört Drucker-Bitmap MOTOR_P.BMP
zu GETRIEBE.SBL gehört Drucker-Bitmap GETRIEBE_P.BMP



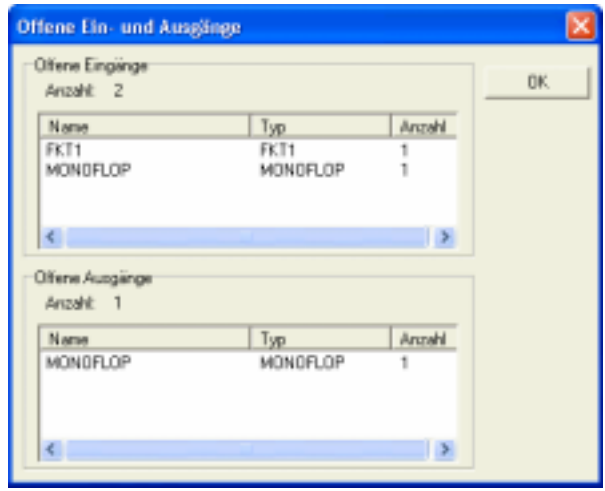
Beispiel für Superblöcke mit anwenderspezifischem Block-Bitmap

Schließlich kann auch für die Darstellung des Superblocks in der Palette *Super* der Systemblock-Toolbar ein Bitmap definiert werden. Dieses hat die Größe von 18x18 Pixeln und den gleichen Namen wie das Drucker-Bitmap, jedoch die Kennung *_T*.

Beispiele: zu MOTOR.SBL gehört Toolbar-Bitmap MOTOR_T.BMP
zu GETRIEBE.SBL gehört Toolbar-Bitmap GETRIEBE_T.BMP

Was sonst noch wissenswert ist

Um sich einen Überblick zu verschaffen, wieviele offene Ein- und Ausgänge das aktuelle System besitzt, dient die Menüfolge DATEI | OFFENE EIN-/AUSGÄNGE... Es erscheint ein Dialog, der jeweils Blocknamen, Blocktyp und die Anzahl der offenen Ein- bzw. Ausgänge auflistet.



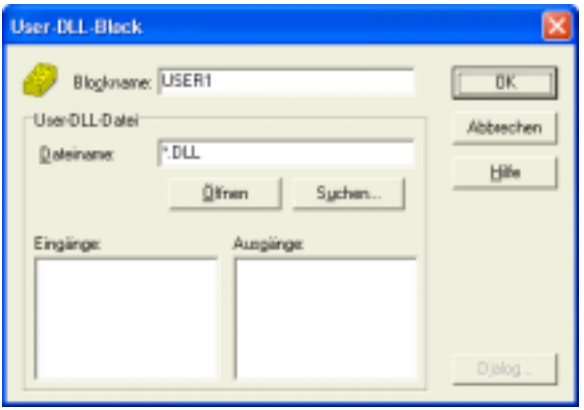
Dialog zur Auflistung offener Ein- und Ausgänge

In einigen Fällen kann es passieren, dass einige Eingänge eines Teilsystems, das zu einem Superblock gruppiert werden soll, offen sind, aber nicht benötigt werden (z. B. Steuereingänge o. ä.) und daher auch nicht als Eingänge des Superblocks auftauchen sollen. In diesen Fällen sollte man diese Eingänge einfach mit einem Konstanten-Block beschalten, der einen geeigneten Wert (in der Regel wird dies 0 sein) aufschaltet.

Benutzerdefinierte Systemblöcke

Das Konzept der User-DLLs

BORIS erlaubt die Programmierung eigener Systemblocktypen, sog. *User-DLL-Blöcke*, auf Basis einer 32-Bit-Windows-DLL. Diese lassen sich mit praktisch jeder 32-Bit-Programmierumgebung wie z. B. Borland Delphi, Borland C++ Builder, Visual Basic oder Visual C++ erstellen. Für den programmierunerfahrenen Anwender steht dazu der optional erhältliche *BORIS-User-DLL-Experte* zur Verfügung.



Parameterdialog eines User-DLL-Blocks (hier noch leer!)

Im folgenden werden zunächst die *Datenschnittstelle* und die *Funktionsschnittstelle* des User-Blocks besprochen, bevor dann anhand von Beispielen unterschiedlicher Komplexität eine tiefergehende Einführung in die Programmierung von User-Blöcken erfolgt.

Hinweis: Alle nachfolgend abgedruckten Programme bzw. Programmsegmente wurden mit Delphi 3 erstellt. Eine Übertragung auf andere Entwicklungsumgebungen oder Programmiersprachen ist aber ohne größere Probleme möglich. Spezielle Hinwese zur Programmierung von User-DLLs unter Microsoft Visual C++ finden Sie im Abschnitt *Hinweise zur Programmierung unter Visual C++*.

Die Datenschnittstelle des User-Blocks

Alle für den Anwender relevanten Daten eines User-Blocks sind in einer Datenstruktur vom Typ *TParameterStruct* bzw. einem Zeiger *PParameterStruct* auf diese Struktur festgelegt. Diese Datenstruktur enthält zunächst die eigentlichen *Blockparameter*. Diese Daten sind in der Regel blockspezifische Konstanten (z. B. Verstärkungsfaktoren, Zeitkonstanten oder ähnliches) und können vom Anwender über den Parameterdialog des User-Blocks geändert werden. Sie werden beim Speichern einer BORIS-Struktur mit dem User-Block abgespeichert und stehen damit nach dem erneuten Einlesen der Datei wieder zur Verfügung. Darüber hinaus können diese Parameter natürlich auch für andere Zwecke - z. B. als Zustandsvariablen, Zwischen- oder Hilfsvariablen irgendwelcher Art, Flags usw. - "missbraucht" werden.

Folgende Parameter stehen zur Verfügung:

- Bis zu 32 Fließkommazahlen (10 Byte, Datentyp *extended* in PASCAL bzw. *long double* in C)
- Bis zu 32 Integer-Zahlen (4 Byte, Datentyp *longint* bzw. *integer* in Pascal bzw. *int* in C)
- Bis zu 32 Schalter- bzw. Byte-Variablen (1 Byte, Datentyp *byte* in Pascal bzw. *char* in C)
- Eine Stringvariable der Länge 256 (z. B. für Dateinamen)

Für die Fließkomma- bzw. Integer-Parameter können Minimal- und Maximalwerte vorgegeben werden, deren Einhaltung dann im Parameterdialog automatisch überprüft wird. Für alle Parameter können bzw. müssen außerdem Namen

vergeben werden, die dann an entsprechender Stelle im Parameterdialog erscheinen. Weitere Parameter können bei Bedarf auch aus einer zusätzlichen Datei gelesen werden. Neben diesen eigentlichen Blockparametern enthält die Datenstruktur *TParameterStruct* einige weitere Variablen, die nur für fortgeschrittene Anwendungen benötigt werden und an späterer Stelle im Rahmen der Beispiele genauer erläutert werden. Zunächst folgt eine knappe Auflistung der Komponenten von *TParameterStruct*.

```
type
  PParameterStruct = ^TParameterStruct;
  TParameterStruct = packed record
    NuE: Byte;                    {Anzahl Fließkomma-Parameter}
    NuI: Byte;                    {Anzahl Integer-Parameter}
    NuB: Byte;                    {Anzahl Schalter-Parameter}
    E: Array[0..31] of Extended;  {Fließkomma-Parameter}
    I: Array[0..31] of LongInt;   {Integer-Parameter}
    B: Array[0..31] of Byte;      {Schalter-Parameter}
    D: Array[0..255] of char;     {Dateiname für optionale Daten}
    EMin: Array[0..31] of Extended; {unt. Grenze Fließkomma-Par.}
    EMax: Array[0..31] of Extended; {obere Grenze Fließkomma-Par.}
    IMin: Array[0..31] of LongInt;  {untere Grenze Integer-Par.}
    IMax: Array[0..31] of LongInt;  {obere Grenze Integer-Par.}
    NaE: Array[0..31,0..40] of char; {Namen Fließkomma-Parameter}
    NaI: Array[0..31,0..40] of char; {Namen Integer-Parameter}
    NaB: Array[0..31,0..40] of char; {Namen Schalter-Parameter}
    UserDataPtr: Pointer;            {Zeiger auf opt. Variablen}
    ParentPtr: Pointer;              {Zeiger auf User-DLL-Block}
    ParentHWnd: Word;                {BORIS-Fensterhandle}
    ParentName: PChar;               {Name des User-DLL-Blocks}
    UserHWindow: Word;               {Benutzerdef. Fensterhandle,
                                      z. B. für Ausgabefenster}
    DataFile: text;                  {Optionale Textdatei}
  end;
```

Struktur der Datenschnittstelle TParameterStruct *in Pascal*

Variable	Bedeutung
NuE, NuI, NuB	Enthält die Anzahl der Fließkomma-, Integer- bzw. Schalterparameter des Blocks. Dabei sind nur die wirklich als Blockparameter benutzten Komponenten zu berücksichtigen, die im Parameterdialog erscheinen sollen, nicht aber die als Hilfsgrößen (z. B. Zustandsvariablen, Flags o. ä.) verwendeten Komponenten der Arrays *E, I* bzw. *B*!
E, I, B	Diese Arrays enthalten die Parameter selbst.
D	Enthält bei Bedarf den Namen einer externen Datei für das Einlesen weiterer Daten.

EMin, EMax	Diese Arrays enthalten die unteren bzw. oberen zulässigen Werte für die Fließkomma-Parameter
IMin, IMax	Diese Arrays enthalten die unteren bzw. oberen zulässigen Werte für die Integer-Parameter.
NaE, NaI, NaB	Arrays mit den Namen der Fließkomma-, Integer- bzw. Schaltervariablen. Jeder Name darf maximal 40 Zeichen aufweisen
UserDataPtr	Zeiger auf optionale Blockvariablen (z. B. Zustandsgrößen, Zwischenwerte oder ähnliches). Hierunter sind in der Regel solche Variablen zu verstehen, die keine Blockparameter sind und damit auch nicht mit dem Block abgespeichert werden müssen.
ParentPtr	Dieser Zeiger zeigt auf den User-DLL-Block. Wird für Anwender-DLLs im allgemeinen nicht benötigt.
ParentHWnd	Handle des BORIS-Hauptfensters. Wichtig bei User-DLLs, die eigene Fenster (z. B. zur Visualisierung) oder Dialoge besitzen.
ParentName	Zeiger auf den Namen des User-DLL-Blocks. Wird i. a. nur benötigt bei User-DLL-Blöcken mit Visualisierungsfunktion
UserHWindow	In dieser Variablen kann z. B. das Fensterhandle eines mit dem User-Block generierten Visualisierungsfensters gespeichert werden. Wird i. a. nur benötigt bei User-DLL-Blöcken mit Visualisierungsfunktion.
DataFile	Textdatei für universelle Zwecke. Kann zusammen mit der Variablen *D* (s. o.) verwendet werden.

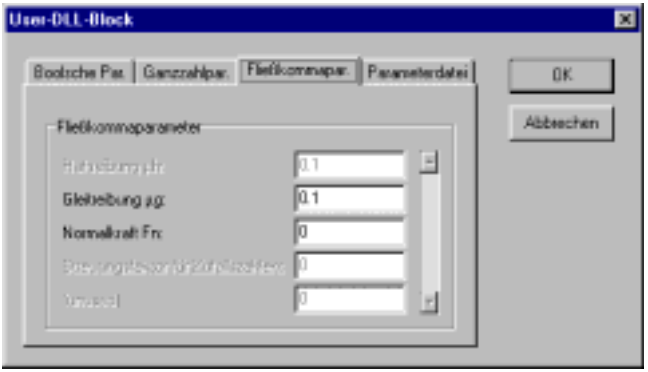
Neben der wichtigsten Datenstruktur *TParameterStruct* müssen in der User-DLL einige weitere Datenstrukturen deklariert sein. Dazu gehört zunächst die Struktur *TDialogEnableStruct* bzw. der zugehörige Zeiger *PDialogEnableStruct*.

```
type
  PDialogEnableStruct=^TDialogEnableStruct;
  TDialogEnableStruct=record
    AllowE: Longint;           { Soll die Eingabe eines Wertes   }
    AllowI: Longint;           { un-/zulässig sein so ist das Bit}
    AllowB: Longint;           { des Allow?-Feldes 0 bzw. 1}
    AllowD: Byte;
  end;
```

Struktur von TDialogEnableStruct *(in Pascal)*

Diese Struktur wird zur Verwaltung des Parameterdialogs benötigt; sie ermöglicht im Zusammenspiel mit der Prozedur *GetDialogEnableStruct* (siehe später) ein bedingtes Aktiv- bzw. Passivsetzen von Dialogelementen (Beispiel: Ein Fließkomma-Parameter soll nur modifizierbar sein, wenn ein bestimmter Schalter im Dialog gesetzt ist). Dazu enthält *TDialogEnableStruct* für jedes der jeweils 32 Eingabefelder eines Variablentyps ein entsprechendes Bit, das gesetzt (Eingabefeld wird aktiv) bzw. gelöscht (Eingabefeld wird passiv) wird. Das Feld *AllowD* legt schließlich fest, ob der Name der optionalen Parameterdatei (Variable *D* aus *TParameterStruct*) aktiv ist.

Nachfolgende Grafik zeigt beispielhaft die Palettenseite *Fließkommaparameter* des Parameterdialogs eines User-DLL-Blocks zur Reibungssimulation, der sich unter dem Namen REIBUNG.DLL im Verzeichnis *UserDLLs* (Quelltext REIBUNG.DPR im Unterverzeichnis *Sources*) befindet. Hier wurden u. a. vier Fließkomma-Parameter definiert, nur der zweite und dritte sind z. Z. aktiv.



Beispiel für den Parameterdialog eines User-Blocks mit vier Fließkomma-Parametern

Die nächsten erforderlichen Datenstrukturen betreffen die Definition der Ein- und/oder Ausgänge des User-Blocks. Zunächst ist die Struktur *TNumberOfInputsOutputs* zu deklarieren. In dieser Struktur sind die Anzahl der Ein- bzw. Ausgänge des Blocks sowie ihre Namen (jeweils max. 40 Zeichen) festgelegt:

```
type
  PNumberOfInputsOutputs = ^TNumberOfInputsOutputs;
  TNumberOfInputsOutputs = packed record
    Inputs  :Byte;                          {Anzahl Eingänge}
    Outputs :Byte;                          {Anzahl Ausgänge}
    NameI : Array[0..49] of String[40];     {Namen der Eingänge}
    NameO : Array[0..49] of String[40];     {Namen der Ausgänge}
  end;
```

Struktur von TNumberOfInputsOutputs *(in Pascal)*

Schließlich bleibt die Deklaration der Datentypen für die spätere Übergabe der aktuellen Blockein- und -ausgangswerte. Dazu dienen die Datentypen *TInputArray* bzw. *TOutputArray*:

```
type
  PInputArray = ^TInputArray;
  TInputArray = packed Array[1..50] of extended;   {Eingangswerte}
  POutputArray = ^TOutputArray;
  TOutputArray = packed Array[1..50] of extended;  {Ausgangswerte}
```

Datentypen TInputArray *bzw.* TOutputArray *(in Pascal)*

Ein weiterer Datentyp - der allerdings nur in sehr seltenen Fällen benötigt wird - ermöglicht Informationen darüber, welche der Blockeingänge tatsächlich mit einem anderen Systemblock verbunden sind:

```
type
  PConnectedArray = ^TConnectedArray;
  TConnectedArray = packed Array[1..50] of byte;
```

Datentyp TConnectedArray *(in Pascal)*

In diesem Array wird bei Aufruf der Funktion *SetEnhancedInformation2* (siehe später) für jeden der Eingänge eine 1 übergeben, falls der entsprechende Eingang eine Verbindung zu einem anderen Block aufweist sowie eine 0, falls der Eingang nicht verbunden ist.

Sollen schließlich bestimmte Blockparameter als Exportparameter freigegeben werden (z. B. um sie einer numerischen Optimierung oder dem Batch-Betrieb zugänglich zu machen), so kommt ein weiterer Datentyp namens *TExportParStruct* hinzu, der wie folgt deklariert ist:

```
type
  PExportParStruct = ^TExportParStruct;
  TExportParStruct = packed record
    ExportParName: array[0..20] of char;
    ExportParType: char;
    ExportParFValue: extended;
    ExportParIValue: integer;
    ExportParBValue: byte;
    ExportParFMin, ExportParFMax: extended;
    ExportParIMin, ExportParIMax: integer;
  end;
```

Datentyp TExportParStruct *(in Pascal)*

Dieser Datentyp dient zur Deklaration eines Exportparameters vom *Fließkomma-*, *Integer-* oder *Boolean*-Typ. Die einzelnen Variablen haben die folgende Bedeutung:

Variable	Bedeutung
ExportParName	Bezeichnung des Parameters (max. 20 Zeichen)
ExportParType	Kennzeichnung des Parametertyps: 'F' = Fließkommaparameter 'I' = Ganzzahlparameter 'B' = boolscher Parameter
ExportParFValue	Wert des Parameters, falls dieser vom Typ *Fließkomma* ist
ExportParIValue	Wert des Parameters, falls dieser vom Typ *Integer* ist
ExportParBValue	Wert des Parameters, falls dieser vom Typ *Boolean* ist
ExportParFMin, *ExportParFMax*	Zulässiger Wertebereich des Parameters, falls dieser vom Typ *Fließkomma* ist
ExportParIMin, *ExportParIMax*	Zulässiger Wertebereich des Parameters, falls dieser vom Typ *Integer* ist

Export-Parameter aus User-DLLs werden völlig analog zu solchen aus "normalen" Systemblöcken behandelt. Der einzige Unterschied besteht darin, dass sie standardmäßig immer freigegeben sind (sofern man nicht in der User-DLL eine entsprechende Aktivierung bzw. Deaktivierung implementiert); die BORIS-Menüoptionen BEARBEITEN | EXPORTPARAMETER | ALLE AKTIVIEREN bzw. BEARBEITEN | EXPORTPARAMETER | ALLE DEAKTIVIEREN haben aus diesem Grund auf die Exportparameter von User-DLLs *keinen* Einfluss.

Damit sind alle erforderlichen Typendeklarationen erläutert. Die Deklaration in C erfolgt analog; nachfolgendes Listing fasst alle Datentypen dafür zusammen.

```
typedef struct{
    char NuE;
    char NuI;
    char NuB;
    long double E[32];
    int  I[32];
    char B[32];
    char D[256];
    long double EMin[32];
    long double EMax[32];
    int  IMin[32];
    int  IMax[32];
    char NaE[32][41];
    char NaI[32][41];
    char NaB[32][41];
    void FAR *UserDataPtr;
```



```
    void FAR *ParentPtr;
    unsigned int ParentHWnd;
    char *ParentName;
    unsigned int UserHWindow;
    FILE *DataFile
  } TParameterStruct, FAR *PParameterStruct;

typedef struct {
    long AllowE;
    long AllowI;
    long AllowB;
    char AllowD;
  } TDialogEnableStruct, FAR *PDialogEnableStruct;

typedef struct {
    char Inputs;
    char Outputs;
    char NameI[50][41];
    char NameO[50][41];
  } TNumberOfInputsOutputs, FAR *PNumberOfInputsOutputs;

typedef long double TInputArray[50];
typedef long double TOutputArray[50];

typedef char TConnectedArray[50];

typedef struct {
    char ExportParName[21];
    char ExportParType;
    long double ExportParFValue;
    int ExportParIValue;
    char ExportParBValue;
    long double ExportParFMin, ExportParFMax;
    int ExportParIMin, ExportParIMax;
  } TExportParStruct, FAR *PExportParStruct
```

Datenschnittstelle des User-Blocks in der Sprache C

Die Funktionsschnittstelle des User-Blocks

Die Funktionsschnittstelle der User-DLL enthält die für den User-Block zu definierenden Funktionen bzw. Prozeduren. Diese unterscheiden sich in solche Funktionen, die zur Verwaltung des User-Blocks bzw. seines Parameterdialogs erforderlich sind (*organisierende* Funktionen) und die Funktionen, die die eigentliche Funktionalität des Blocks beschreiben, d. h. während der Simulation bzw. unmittelbar davor oder danach aufgerufen werden (*simulierende* Funktionen). Folgende Auflistung gibt zunächst eine Übersicht über alle erlaubten Funktionen mit ihren Parametern, bevor dann die Bedeutung aller Funktionen im Detail erläutert wird. Alle Funktionen sind als *stdcall* zu deklarieren.

Organisierende Funktionen:

```
Procedure IsUserDLL32;

Procedure GetParameterStruct(D:PParameterStruct);

Procedure GetDialogEnableStruct(D:PDialogEnableStruct;
                                D2:PParameterStruct)

Procedure GetNumberOfInputsOutputs(D:PNumberOfInputsOutputs);
Procedure GetNumberOfInputsOutputs2(D:PNumberOfInputsOutputs;
                                    ParameterFileName: PChar;
                                    UserDataPtr: Pointer;
                                    var UserHWindow: THandle);

Procedure InitUserDLL(D:PParameterStruct);
Procedure DisposeUserDLL(D:PParameterStruct);

Procedure InitUserData(D: PParameterStruct);
Procedure DisposeUserData(D: PParameterStruct);

Procedure DialogOK(D:PParameterStruct);

Procedure ShowWindowDLL(D: PParameterStruct);
Procedure HideWindowDLL(D: PParameterStruct);

Function  SetInputChar: PChar;
Function  SetOutputChar: PChar;

Function  GetDLLName: PChar;

Procedure WriteToFile(AFileHandle: word; D: PParameterStruct);
Procedure ReadFromFile(AFileHandle: word; D: PParameterStruct);
Function  NumberOfLinesInSystemFile: word;

Function  WriteParToDocFileDLL(D: PParameterStruct): integer;

Procedure CallParameterDialogDLL(D1: PParameterStruct;
                                 D2: PNumberOfInputsOutputs);

Function  ExportParCountDLL(D1: PParameterStruct;
                            D2: PNumberOfInputsOutputs): integer;
Procedure GetExportParDLL(D1: PParameterStruct;
                          D2: PNumberOfInputsOutputs;
                          ParIndex: integer; D3: PExportParStruct);
Procedure SetExportParDLL(D1: PParameterStruct;
                          D2: PNumberOfInputsOutputs;
                          ParIndex: integer; D3: PExportParStruct);
```

Simulierende Funktionen:

```
Function  CanSimulateDLL(D:PParameterStruct):Integer;
Function  CanSimulateDLL2(T, DeltaT:Extended;
                          D:PParameterStruct):Integer;

Procedure InitSimulationDLL(D:PParameterStruct;
                            Inputs:PInputArray;
                            Outputs:POutputArray);
```



```
Procedure InitSimulationDLL2(D:PParameterStruct;
                             DeltaT:Extended;
                             Inputs:PInputArray;
                             Outputs:POutputArray);

Procedure SimulateDLL(T:Extended; D:PParameterStruct;
                      Inputs:PInputArray;
                      Outputs:POutputArray);
Procedure SimulateDLL2(T, DeltaT:Extended; D:PParameterStruct;
                       Inputs:PInputArray;
                       Outputs:POutputArray);
Function SimulateDLL3(T, DeltaT:Extended; D:PParameterStruct;
                      Inputs:PInputArray;
                      Outputs:POutputArray): integer;

Procedure EndSimulationDLL;
Procedure EndSimulationDLL2(D: PParameterStruct);

Procedure SetEnhancedInformation(DeltaT, TSimu: extended;
                                 D: PParameterStruct);
Procedure SetEnhancedInformation2(DeltaT, TSimu: extended;
                                  D1: PConnectedArray;
                                  D2: PParameterStruct);

Function HasDelayDLL(D: PParameterStruct): integer;
```

Bei der nachfolgenden Beschreibung der einzelnen Funktionen ist jeweils sowohl die Pascal-Signatur (oben) als auch die C-Signatur angegeben.

IsUserDLL32

Signatur

```
procedure IsUserDLL32; export stdcall;
```

```
void _export _stdcall IsUserDLL32
```

Funktion Diese Dummy-Funktion wird lediglich exportiert, um die DLL als BORIS-32-Bit-User-DLL zu kennzeichnen. Der eigentliche Funktionsrumpf kann leer bleiben.

GetParameterStruct

Signatur

```
procedure GetParameterStruct(D:PParameterStruct);
                               export stdcall;
```

```
void _export _stdcall GetParameterStruct(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion In *GetParameterStruct* legen Sie die Blockparameter sowie die Daten zum Initialisieren des Parameterdialogs fest. Es werden Namen für Parameter vergeben, die Parameter selber werden initialisiert, obere und untere Eingabegrenzen eines Parameters festgelegt und letztlich wird die Anzahl der Parameter angegeben.

Beispiel:

Nachfolgendes Beispiel definiert einen User-Block mit vier Fließkomma-Parametern mit den Namen *Haftreibung*, *Gleitreibung*, *Normalkraft*, *Streuungsfaktor* sowie zwei Schalter-Parametern mit den Namen *Reale Reibung* bzw. *Stick and Slip aus Datei*. Außerdem wird eine zusätzliche Parameterdatei mit der Extension *SIM* zugelassen.

```
procedure GetParameterStruct(D:PParameterStruct); export stdcall;
var i : Integer;
begin
  D^.NuE:=4;  {Vier Fließkomma-Parameter}
  D^.NuI:=0;  {Keine Integer-Parameter}
  D^.NuB:=2;  {Zwei Schalter-Parameter}
  StrPCopy(D^.D,'*.sim');  {Dateien mit Endung SIM zulassen}
  {Initialisierung der verwendeten Daten}
  for i:=0 to 1 do begin
     D^.E[i]:=0;    {Fließkomma-Parameter 0 und 1 zu 0 initialisieren}
     D^.EMin[i]:=0; {Untere Grenze für Fließkomma-Par. 0 und 1 auf 0}
     D^.EMax[i]:=1; {Obere Grenze für Fließkomma-Par. 0 und 1 auf 1}
     D^.B[i]:=0;    {Schalter-Parameter zu 0 initialisieren}
  end;
  D^.E[2]:=0;          {Fließkomma-Parameter 2 zu 0 initialisieren}
  D^.EMin[2]:=0;       {Untere Grenze von Fließkomma-Par. 2 auf 0}
  D^.EMax[2]:=100000;  {Obere Grenze von Fließkomma-Par. 2 auf 100000}
  D^.E[3]:=0;          {Fließkomma-Parameter 3 zu 0 initialisieren}
  D^.EMin[3]:=0;       {Untere Grenze von Fließkomma-Par. 3 auf 0}
  D^.EMax[3]:=1;       {Untere Grenze von Fließkomma-Par. 3 auf 1}
  {Namensgebung max. 40 Zeichen !}
  strPCopy(D^.NaE[0],'Haftreibung:');     {Name für Fließkomma-Par. 0}
  strPCopy(D^.NaE[1],'Gleitreibung:');    {Name für Fließkomma-Par. 1}
  strPCopy(D^.NaE[2],'Normalkraft:');     {Name für Fließkomma-Par. 2}
  strPCopy(D^.NaE[3],'Streuungsfaktor:'); {Name für Fließkomma-Par. 3}
  strPCopy(D^.NaB[0],'Reale Reibung');    {Name für Schalter-Par. 0}
  strPCopy(D^.NaB[1],'"Stick and Slip" aus Datei'); {N. f. Sch.-P. 1}
end;
```

Beispiel für die Verwendung der Funktion GetParameterStruct

GetDialogEnableStruct

Signatur

```
procedure getDialogEnableStruct(D:PDialogEnableStruct;
                                D2:PParameterStruct);
                                export stdcall;

void _export _stdcall GetDialogEnableStruct
                      (PDialogEnableStruct D,
                       PParameterStruct D2)
```

Parameter *D* ist ein Zeiger auf die Struktur *TDialogEnableStruct.*

D2 ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion *GetDialogEnableStruct* legt die zugänglichen bzw. gesperrten (grau dargestellten) Elemente des Parametrierungsdialogs fest, d. h. welche Dialogelemente vom Anwender in welcher Situation

angewählt werden können und welche nicht. Die Funktion wird bei Aufruf des Dialogs sowie *nach jeder Eingabe* aufgerufen, die vom Anwender im Dialog vorgenommen wird. Somit kann sie ggf. unmittelbar auf diese reagieren.

Beispiel:

Bei einem User-Block mit Fließkomma- und Schalter-Parametern sollen alle Fließkomma- und Schalter-Parameter jederzeit anwählbar sein. Der Dateiname für die optionale Parameterdatei soll aber nur zugänglich sein, wenn beide Schalter-Parameter gesetzt sind.

```
procedure GetDialogEnableStruct(D:PDialogEnableStruct;
                                D2:PParameterStruct); export stdcall;
begin
  D^.AllowE:=$FFFFFFFF;  {alle Fließkomma-Parameter zugänglich}
  D^.AllowB:=$FFFFFFFF   {alle Schalter-Parameter zugänglich};
  {Dateiname nur zugänglich, wenn beide Schalter-Parameter
  gesetzt sind!}
  D^.AllowD:= Byte(D2^.B[0] and D2^.B[1]);
end;
```

Beispiel für die Verwendung der Funktion GetDialogEnableStruct

Natürlich ist es auch möglich, z. B. nur ein Dialogelement vom gesperrten in den zugänglichen Zustand (oder umgekehrt) zu überführen und die anderen unbeeinflusst zu lassen. Soll z. B. der dritte Fließkomma-Parameter (also Fließkomma-Parameter 2) aktiviert werden, alle anderen Fließkomma-Parameter aber ihren aktuellen Zustand beibehalten, so lautet die entsprechende Anweisung

```
...
AllowE := AllowE or $00000004;   {Fließkomma-Parameter 2 freigeben}
...
```

GetNumberOfInputsOutputs

Signatur

```
procedure GetNumberOfInputsOutputs
          (D:PNumberOfInputsOutputs); export stdcall;

void _export _stdcall GetNumberOfInputsOutputs
                      (PNumberOfInputsOutputs D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TNumberOfInputsOutputs.*

Funktion In *GetNumberOfInputsOutputs* werden Anzahl und Namen der Ein- und Ausgänge des Blocks festgelegt. Die Namen werden später in den Listboxen *Eingangsvariable(n)* und *Ausgangsvariable(n)* des User-DLL-Dialogs angezeigt.

Beispiel:

Folgendes Listing zeigt die Realisierung der Funktion für einen User-Block mit zwei Eingängen und einem Ausgang.

```
procedure GetNumberOfInputsOutputs(D:PNumberOfInputsOutputs); export
                                                              stdcall;
begin
  D^.Inputs:=2;   { Der Block soll zwei Eingänge und}
  D^.Outputs:=1;  {             einen Ausgang haben!}
  {Namensgebung der Ein- und Ausgänge}
  StrPCopy(D^.NameI[0],'Sollwert');    {Name für ersten Eingang}
  StrPCopy(D^.NameI[1],'Istwert');     {Name für zweiten Eingang}
  StrPCopy(D^.NameO[0],'Stellgröße');  {Name für Ausgang}
end;
```

Beispiel für die Verwendung der Funktion GetNumberOfInputsOutputs

GetNumberOfInputsOutputs2

Signatur

```
Procedure GetNumberOfInputsOutputs2
           (D:PNumberOfInputsOutputs;
            ParameterFileName: PChar;
            UserDataPtr: Pointer;
            var UserHWindow: THandle); export stdcall

void _export _stdcall GetNumberOfInputsOutputs2
                 (PNumberOfInputsOutputs D,
                  char* ParameterFileName,
                  void* UserDataPtr,
                  int* UserHWindow)
```

Parameter *D* ist ein Zeiger auf die Struktur *TNumberOfInputsOutputs.*

ParameterFileName ist ein Zeiger auf das Feld *D* von *TParameterStruct.*

UserDataPtr ist ein Zeiger auf das Feld *UserDataPtr* von *TParameterStruct.*

UserHWindow ist ein Zeiger auf das Feld *UserHWindow* von *TParameterStruct.*

Funktion Diese Funktion kann alternativ zu *GetNumberOfInputsOutputs* benutzt werden, wenn die Anzahl der Ein- und Ausgänge variabel sein soll.

Beispiel:

Folgendes Listing zeigt die Realisierung der Funktion für einen User-Block, bei dem die Anzahl der Ein- und Ausgänge aus der Datei *ParameterFileName* gelesen werden sollen.


```
Procedure GetNumberOfInputsOutputs2(D:PNumberOfInputsOutputs;
           ParameterFileName: PChar; UserDataPtr: Pointer;
           var UserHWindow: THandle); export stdcall
var ParFile: TextFile;
    nIn, nOut, i: integer
begin
  AssignFile(ParFile, ParameterFileName);  //Dateinamen zuweisen
  ResetFile(ParFile);                      //Datei öffnen
  readln(ParFile, nIn, nOut);              //Anzahl Ein-/Ausgänge einlesen
  D^.Inputs  := nIn;                       //...und dem Block zuweisen
  D^.Outputs := nOut;
  {Namensgebung der Ein- und Ausgänge}
  for i:=1 to nIn  do StrPCopy(D^.NameI[i-1], 'Eingang' + IntToStr(i));
  for i:=1 to nOut do StrPCopy(D^.NameO[i-1], 'Ausgang' + IntToStr(i));
end;
```

Beispiel für die Verwendung der Funktion GetNumberOfInputsOutputs2

InitUserDLL
DisposeUserDLL

Signatur

```
procedure InitUserDLL(D:PParameterStruct);
                                        export stdcall;
procedure DisposeUserDLL(D:PParameterStruct);
                                        export stdcall;

void _export _stdcall InitUserDLL(PParameterStruct D)
void _export _stdcall DisposeUserDLL(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion *InitUserDLL* wird von BORIS unmittelbar nach der Initialisierung eines User-Blocks aufgerufen. Sie können diese Funktion beispielsweise nutzen, um dynamischen Speicher für Daten anzulegen, die Sie nicht in der Parameterstruktur *TParameterStruct* des Blocks unterbringen wollen oder können. Dies werden in der Regel z. B. Zustandsgrößen, Zwischenwerte irgendwelcher Art oder umfangreichere Parameterstrukturen sein, die aus einer Parameterdatei gelesen wurden. Den Zeiger auf den hier angelegten dynamischen Speicher können Sie im Parameter *UserDataPtr* von *TParameterStruct* ablegen.

Ein anderer Anwendungsfall für diese Funktion sind User-Blöcke mit Visualisierungsfenster. Bei solchen Anwendungen kann das Visualisierungsfenster des User-Blocks in dieser Funktion generiert werden (siehe dazu das Beispiel *Füllstandsregelung* an späterer Stelle!).

Das Gegenstück zu *InitUserDLL* stellt *DisposeUserDLL* dar. Diese Funktion wird unmittelbar vor der Freigabe eines User-Blocks aufgerufen. Der in *InitUserDLL* angelegte Speicher muss

daher in *DisposeUserDLL* wieder freigegeben werden.

Werden die erforderlichen Daten nicht während der gesamten "Lebensdauer" des User-Blocks, sondern nur während der eigentlichen Simulation selbst benötigt (dies wird in der Regel häufiger der Fall sein), können statt der Funktionen *InitUserDLL* und *DisposeUserDLL* auch die Funktionen *InitUserData* und *DisposeUserData* (werden nachfolgend beschrieben) benutzt werden.

Die Verwendung von *InitUserDLL* und *DisposeUserDLL* ist in der Regel nur in komplexeren User-Blöcken erforderlich.

Beispiel:

Es soll ein User-Block erstellt werden, der neben den Blockparametern eine 20x20-Matrix vom Typ *Extended* sowie einen Vektor der Dimension 50 vom Typ *Integer* benötigt. Folgendes Listing zeigt die entsprechende Verwendung der Funktionen *InitUserDLL* und *DisposeUserDLL*.

```
...
...
type
  {Deklaration des Datentyps für die User-Daten}
  PUserData = ^TUserData;
  TUserData = record
    Matrix: array[1..20, 1..20] of extended;
    Vektor: array[1..50] of integer;
  end;


procedure InitUserDLL(D: PParameterStruct); export stdcall;
begin
  ...
  ...
  {Speicher für User-Daten anfordern}
  GetMem(D^.UserDataPtr, SizeOf(TUserData));
  ...
  ...
end {of InitUserDLL};


procedure DisposeUserDLL(D: PParameterStruct); export stdcall;
begin
  ...
  ...
  {Speicher für User-Daten wieder freigeben}
  FreeMem(D^.UserDataPtr, SizeOf(TUserData));
  ...
  ...
end {of DisposeUserDLL};
...
...
```

Beispiel für die Verwendung von InitUserDLL *und* DisposeUserDLL

Soll während der Simulation dann z. B. auf ein bestimmtes Matrixelement zugegriffen werden, lautet die entsprechende Pascal-Anweisung:


```
...
...
{2. Zeile, 3. Spalte der Matrix auf 5 setzen}
PUserData(UserDataPtr)^.Matrix[2, 3] := 5;
...
...
```

Zugriff auf User-Daten

InitUserData
DisposeUserData

Signatur

```
procedure InitUserData(D:PParameterStruct);
                                          export stdcall;
procedure DisposeUserData(D:PParameterStruct);
                                          export stdcall;

void _export _stdcall InitUserData(PParameterStruct D)
void _export _stdcall DisposeUserData(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion *InitUserData* und *DisposeUserData* entsprechen von ihrem Anwendungsbereich her den zuvor beschriebenen Funktionen *InitUserDLL* und *DisposeUserDLL*, werden aber nicht bei der Initialisierung bzw. Freigabe des User-Blocks, sondern unmittelbar vor bzw. nach der Simulation (d. h. direkt vor *InitSimulationDLL* bzw. nach *EndSimulationDLL*) aufgerufen. Sie eignen sich damit besonders für die Verwaltung von Daten, die nur während der Simulation selbst benötigt werden (siehe dazu das Beispiel *Kennfeld-DLL* an späterer Stelle). Auch diese Funktionen sind für die Realisierung einfacher DLLs nicht erforderlich.

DialogOK

Signatur

```
procedure DialogOK(D:PParameterStruct);
                                        export stdcall;

void _export _stdcall DialogOK(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion *DialogOK* ist in der Regel nur bei User-Blöcken mit Visualisierungsfenster erforderlich und dort auch nur in wenigen Fällen. Diese Funktion wird von BORIS aufgerufen, wenn der Anwender den Parameterdialog des User-Blocks über den *OK*-Schalter verlassen hat. Sie kann beispielsweise benutzt werden, um den Inhalt des Visualisierungsfensters an die geänderten Blockparameter anzupassen.

Beispiel:

Folgendes Listing zeigt ein Beispiel für die Anwendung von *DialogOK*.

```
procedure DialogOK(D:PParameterStruct); export stdcall;
begin
  {Nachdem der Parameterdialog über OK verlassen wurde,
  soll das Visualisierungsfenster, dessen Handle in
  UserHWindow liegt, aufgefrischt werden!}
  InvalidateRect(D^.UserHWindow, nil, true);
end;
```

Beispiel für die Anwendung von DialogOK

ShowWindowDLL
HideWindowDLL

Signatur

```
Procedure ShowWindowDLL(D: PParameterStruct);
                                        export stdcall;
Procedure HideWindowDLL(D: PParameterStruct);
                                        export stdcall;

void _export _stdcall ShowWindowDLL(PParameterStruct D)
void _export _stdcall HideWindowDLL(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion Diese Funktionen werden nur bei User-DLLs mit Visualisierungsfenstern benötigt. Sie können benutzt werden, um das Visualisierungsfenster anzuzeigen bzw. zum Symbol zu verkleinern. Die Funktionen werden von BORIS aufgerufen, wenn der Anwender die Option OPTIONEN | ALLE ANZEIGEFENSTER ZEIGEN bzw. OPTIONEN | ALLE ANZEIGEFENSTER VERBERGEN anwählt.

Beispiel:

Folgendes Listing zeigt ein Anwendungsbeispiel für die beiden Funktionen:

```
...
procedure ShowWindowDLL(D: PParameterStruct); export stdcall;
begin
  {Anzeigefenster in Normalgröße anzeigen}
  ShowWindow(D^.UserHWindow, sw_Normal);
end;

procedure HideWindowDLL(D: PParameterStruct); export stdcall;
begin
  {Anzeigefenster zum Symbol verkleinern}
  ShowWindow(D^.UserHWindow, sw_Minimize);
end;
```

Beispiel für die Anwendung von ShowWindowDLL *bzw.* HideWindowDLL

SetInputChar
SetOutputChar

Signatur

```
Function SetInputChar: PChar; export stdcall;
Function SetOutputChar: PChar; export stdcall;

char* _export _stdcall SetInputChar(void)
char* _export _stdcall SetOutputChar(void)
```

Rückgabewert Der Rückgabewert enthält einen String, dessen Zeichen die Beschriftung der einzelnen Blockeingänge bzw. Blockausgänge festlegen.

Funktion *SetInputChar* bzw. *SetOutputChar* können benutzt werden, um die standardmäßige Beschriftung der Blockeingänge bzw. -ausgänge des User-Blocks in der Strukturdarstellung zu ändern. In der Regel wird der Eingang eines Blocks mit *E* (bei nur einem Eingang) und der Ausgang mit *A* (bei nur einem Ausgang) beschriftet, bei mehreren Ein- bzw. Ausgängen werden diese durchnumeriert. Durch Implementierung von *SetInputChar* bzw. *SetOutputChar* kann für jeden Ein- bzw. Ausgang stattdessen ein benutzerdefiniertes Zeichen vorgegeben werden.

Beispiel:

Bei einem User-Block mit zwei Eingängen und einem Ausgang sollen die Eingänge mit *w* und *x* und der Ausgang mit *y* beschriftet werden. Nachfolgendes Listing zeigt die entsprechende Implementierung der Funktionen *SetInputChar* und *SetOutputChar*.

```
Function SetInputChar: PChar; export stdcall;
begin
  SetInputChar := 'wx'; {Eingang 1 mit "w", Eingang 2 mit "x" beschriften}
end;

Function SetOutputChar: PChar; export stdcall;
begin
  SetOutputChar := 'y'; {Ausgang mit "y" beschriften}
end;
```

Beispiel für die Anwendung von SetInputChar *und* SetOutputChar

GetDLLName

Signatur

```
Function GetDLLName: PChar; export stdcall;

char _export _stdcall GetDLLName(void)
```

Rückgabewert Bezeichnung des User-DLL-Blocks

Funktion Über diese Funktion kann dem User-DLL-Block ein Name gegeben werden, der dann im Untermenü SYSTEMBLÖCKE | USER-DLL-BLÖCKE bzw. im ToolTip-Fenster der Palettenseite *User* der Systemblock-Toolbar erscheint.

Beispiel:

Nachfolgend definierter User-DLL-Block erhält die Bezeichnung *Riesen-Display*.

```
function GetDLLName: PChar; export stdcall;
begin
  GetDLLName := 'Riesen-Display';
end;
```

Beispiel für die Realisierung von GetDLLName

WriteParToDocFileDLL

Signatur

```
Function WriteParToDocFileDLL(D: PParameterStruct;
                    Count: integer; s: PChar): integer;
                                        export stdcall;

int _export _stdcall WriteParToDocFileDLL
               (PParameterStruct D, Int Count, Char s)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Count ist der Zähler für die auszugebenden Parameter.

s ist der auszugebende Text für den Parameter

Rückgabewert Ein Rückgabewert von 0 gibt an, dass keine weiteren Parameter mehr ausgegeben werden sollen. Bei jedem anderen Rückgabewert wird die Funktion nochmals von BORIS mit einem um eins erhöhten Wert von *Count* aufgerufen.

Funktion Diese Funktion kann benutzt werden, um bei der Dokument-Generierung von BORIS zusätzliche, benutzerdefinierte Blockparameter in die Dokumentdatei auszugeben. Dazu ruft BORIS die Funktion beginnend mit *Count* = 0 so oft auf, bis diese den Rückgabewert 0 liefert. Bei jedem Aufruf liefert die Funktion in *s* den auszugebenden String für den jeweiligen Parameter.
Hinweis: Werden nur die Standardparameter aus *TParameter-Struct* benutzt (also die Arrays *E*, *I* und *B*), so ist diese Funktion nicht erforderlich, da die Standardparameter von BORIS automatisch in die Dokumentdatei übernommen werden!

Beispiel:

Nachfolgendes Listing zeigt die Implementierung der Funktion für den Fall, dass in *UserDataPtr* drei benutzerdefinierte Fließkommaparameter (*a*, *b* und *c*) angelegt wurden, die in die Dokumentdatei geschrieben werden sollen.

```
function WriteParToDocFileDLL(D: PParameterStruct; count: integer;
                                        s: PChar): integer; export stdcall;
begin
  case Count of
    0: begin
         StrPCopy(s, 'a = ' + FloatToStr(D^.UserDataPtr.a) + #13#10);
         Result := 1;  // Es folgen weitere Parameter!
       end;
    1: begin
         StrPCopy(s, 'b = ' + FloatToStr(D^.UserDataPtr.b) + #13#10);
         Result := 1;  // Es folgen weitere Parameter!
       end;
    2: begin
         StrPCopy(s, 'c = ' + FloatToStr(D^.UserDataPtr.c) + #13#10);
         Result := 0;  // Parameter war letzter Parameter!
       end;
  end;
```

Beispiel für die Realisierung von WriteParToDocFileDLL

CallParameterDialogDLL

Signatur

```
Procedure CallParameterDialogDLL(D1: PParameterStruct;
          D2: PNumberOfInputsOutputs); export stdcall;

void _export _stdcall CallParameterDialogDLL
     (PParameterStruct D1, PNumberOfInputsOutputs D2)
```

Parameter *D1* ist ein Zeiger auf die Struktur *TParameterStruct.*

D2 ist ein Zeiger auf die Struktur *TNumberOfInputsOutputs.*

Funktion Diese Funktion ermöglicht den Aufruf eines benutzerdefiniertes Parameterdialogs anstelle des Standard-Dialogs.

Beispiel:

Nachfolgendes Listing aus der Beispiel-DLL DLLDEMO2 zeigt den Aufruf eines benutzerdefinierten Dialogs in DELPHI.

```
procedure CallParameterDialogDLL(D1:PParameterStruct;
                                 D2:PNumberOfInputsOutputs);export stdcall;
(* Stellt den benutzerdefinierten Parameterdialog dar *)
var s: string;
    Code: integer;
begin
  Form1 := TForm1.Create(Application);
  Str(D1^.E[0]:10, s);
  Form1.Edit1.Text := s;
  Form1.Checkbox1.Checked := D1^.B[0] > 0;
  if Form1.ShowModal = mrOk then begin
    Val(Form1.Edit1.Text, D1^.E[0], Code);
```

```
    D1^.B[0] := ord(Form1.Checkbox1.Checked);
  end;
  Form1.Free;
end;
```

Beispiel für die Realisierung von CallParameterDialogDLL

ExportParCountDLL

Signatur

```
Function ExportParCountDLL(D1: PParameterStruct;
                    D2: PNumberOfInputsOutputs): integer;
                                          export stdcall;

int _export _stdcall ExportParCountDLL
     (PParameterStruct D1, PNumberOfInputsOutputs D2)
```

Parameter *D1* ist ein Zeiger auf die Struktur *TParameterStruct.*

D2 ist ein Zeiger auf die Struktur *TNumberOfInputsOutputs.*

Rückgabewert Der Rückgabewert gibt die Anzahl der vom Block exportierten Parameter an.

Funktion Diese Funktion ermöglicht zusammen mit den weiter unten erläuterten Funktionen *GetExportParDLL* und *SetExportParDLL* den Export von Parametern aus der User-DLL. Sollen keine Parameter exportiert werden, brauchen diese Funktionen in der DLL nicht exportiert zu werden.

Beispiel:

Nachfolgendes Listing zeigt die Realisierung dieser Funktion für einen Block, aus dem zwei Parameter exportiert werden sollen, in DELPHI.

```
function ExportParCountDLL(D1: PParameterStruct;
                        D2: PNumberOfInputsOutputs): integer;
                                                  export stdcall;
begin
  Result := 2; // Zwei Parameter exportieren!
end;
```

Beispiel für die Realisierung von ExportParCountDLL

GetExportParDLL

Signatur

```
Procedure GetExportParDLL(D1: PParameterStruct;
                          D2: PNumberOfInputsOutputs;
                          ParIndex: integer;
                          D3: PExportParStruct);
                                       export stdcall;

void _export _stdcall GetExportParDLL
       (PParameterStruct D1, PNumberOfInputsOutputs D2,
```



```
                    int ParIndex, PExportParStruct D3)
```

Parameter *D1* ist ein Zeiger auf die Struktur *TParameterStruct.*

D2 ist ein Zeiger auf die Struktur *TNumberOfInputsOutputs.*

ParIndex ist der Index des Exportparameters (beginnend bei 1)

D3 ist ein Zeiger auf die Struktur *TExportParStruct.*

Funktion Diese Funktion liefert die von BORIS benötigten Informationen über den über *ParIndex* spezifizierten Exportparameter. Dieser wird über den Zeiger *D3* an BORIS zurückgeliefert.

Beispiel:

Nachfolgendes Listing zeigt die Realisierung dieser Funktion für einen Block, aus dem zwei Parameter (beide vom Typ *Fließkomma*) exportiert werden sollen, in DELPHI.

```
procedure GetExportParDLL(D1:PParameterStruct; D2:PNumberOfInputsOutputs;
                          ParIndex: integer; D3: PExportParStruct);
                                                          export stdcall;
begin
  // Beispiel für den Export von zwei Fließkomma-Parametern.
  // Hier werden beispielhaft die Fließkomma-Parameter E[0] und E[3]
  // aus der Struktur TParameterStruct (D1) exportiert.
  case ParIndex of
    // Erster Exportparameter des Blocks
    1: with D3^ do begin
         ExportParName   := 'Verstärkung K';  // Bezeichnung des ersten
                                              // Parameters
         ExportParType   := 'F';              // Parameter ist vom Typ
                                              // Fließkomma
         ExportParFValue := D1^.E[0];         // Es soll sich um den ersten
                                              // Fließkommaparameter aus
                                              // TParameterStruct handeln!
         ExportParFMin   := -1E30;            // Kleinster zulässiger Wert
                                              // des Parameters
         ExportParFMax   := +1E30;            // Größter zulässiger Wert
                                              // des Parameters
       end;
    // Zweiter Exportparameter des Blocks
    2: with D3^ do begin
         ExportParName   := 'Dämpfung D';     // Bezeichnung des zweiten
                                              // Parameters
         ExportParType   := 'F';              // Parameter ist vom Typ
                                              // Fließkomma
         ExportParFValue := D1^.E[3];         // Es soll sich um den
                                              // vierten Fließkommparameter
                                              // aus TParameterStruct
                                              // handeln!
         ExportParFMin   := -1E30;            // Kleinster zulässiger Wert
                                              // des Parameters
         ExportParFMax   := +1E30;            // Größter zulässiger Wert
                                              // des Parameters
       end;
  end;
end;
```

Beispiel für die Realisierung von GetExportParDLL

SetExportParDLL

Signatur

```
Procedure SetExportParDLL(D1: PParameterStruct;
                          D2: PNumberOfInputsOutputs;
                          ParIndex: integer;
                          D3: PExportParStruct);
                                        export stdcall;

void _export _stdcall SetExportParDLL
       (PParameterStruct D1, PNumberOfInputsOutputs D2,
        int ParIndex, PExportParStruct D3)
```

Parameter *D1* ist ein Zeiger auf die Struktur *TParameterStruct.*

D2 ist ein Zeiger auf die Struktur *TNumberOfInputsOutputs.*

ParIndex ist der Index des Exportparameters (beginnend bei 1)

D3 ist ein Zeiger auf die Struktur *TExportParStruct.*

Funktion Diese Funktion stellt das Gegenstück zu *GetExportParDLL* dar. Sie wird von BORIS aufgerufen, wenn dem über *ParIndex* spezifizierten Exportparameter ein neuer Wert zugewiesen werden soll. Dieser wird über den Zeiger *D3* übergeben.

Beispiel:

Nachfolgendes Listing zeigt die Realisierung dieser Funktion für einen Block, aus dem zwei Parameter (beide vom Typ *Fließkomma*) exportiert werden sollen, in DELPHI (vgl. Listing bei *GetExportParDLL*).

```
procedure SetExportParDLL(D1:PParameterStruct; D2:PNumberOfInputsOutputs;
                          ParIndex: integer; D3: PExportParStruct);
                                                      export stdcall;
begin
  // Beispiel für den Export von zwei Fließkomma-Parametern.
  // Hier werden beispielhaft die Fließkomma-Parameter E[0] und E[3]
  // aus der Struktur TParameterStruct (D1) exportiert.
  case ParIndex of
    // Erster Exportparameter des Blocks
    1: with D3^ do begin
         D1^.E[0] := ExportParFValue;  // Es soll sich um den ersten
                                       // Fließkommaparameter aus
                                       // TParameterStruct handeln!
       end;
    // Zweiter Exportparameter des Blocks
    2: with D3^ do begin
         D1^.E[3] := ExportParFValue;  // Es soll sich um den vierten
                                       // Fließkommaparameter aus
                                       // TParameterStruct handeln!
       end;
  end;
end;
```

Beispiel für die Realisierung von SetExportParDLL

WriteToFile
ReadFromFile
NumberOfLinesInSystemFile

Signatur

```
Procedure WriteToFile(AFileHandle: word;
            D: PParameterStruct); export stdcall
Procedure ReadFromFile(AFileHandle: word;
            D: PParameterStruct); export stdcall
Function  NumberOfLinesInSystemFile: word;
                                    export stdcall

void _export _stdcall WriteToFile
     (unsigned short AFileHandle, PParameterStruct D)
void _export _stdcall ReadFromFile
     (unsigned short AFileHandle, PParameterStruct D)

int _export _stdcall NumberOfLinesInSystemFile(void);
```

Parameter *AFileHandle* ist das Handle der Systemdatei.

D ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion Diese drei Funktionen dienen zur Speicherung von blockspezifischen Parametern, die über die 32 Fließkomma-, Ganzzahl- und Schalterparameter hinausgehen. Sie sind daher nur in seltenen Fällen notwendig. Die User-DLL BIGDIS32.DLL im Verzeichnis *UserDLLs* demonstriert die Anwendung der drei Funktionen.

CanSimulateDLL

Signatur

```
function CanSimulateDLL(D:PParameterStruct):Integer;
                                        export stdcall;

int _export _stdcall CanSimulateDLL(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Rückgabewert Der Rückgabewert '0' zeigt an, dass nicht simuliert werden kann, der Rückgabewert von '1', dass dieser Block simulationsbereit ist.

Funktion *CanSimulateDLL* wird von BORIS vor Beginn der Simulation (noch vor *InitSimulationDLL*) aufgerufen, um zu überprüfen, ob der User-Block simuliert werden kann. Mittels dieser Funktion ist es also möglich, unter bestimmten Bedingungen (beispielsweise, wenn der Block eine Parameterdatei benötigt und für diese ein nicht existierender Dateiname angegeben wurde) eine Simulation zu verhindern, indem der Rückgabewert auf 0 gesetzt wird. In den meisten Fällen sind derartige Überprüfungen nicht notwendig und der Rückgabewert kann auf 1 gesetzt werden.

Beispiel:

Folgende Listings zeigen zwei Realisierungsmöglichkeiten für die Funktion *CanSimulateDLL*.

```
Function CanSimulateDLL(D: PParameterStruct): integer; export stdcall;
begin
  CanSimulateDLL := 1; {Simulation immer zulässig!}
end;
```

```
Function CanSimulateDLL(D: PParameterStruct): integer; export stdcall;
begin
  {Die Simulation soll nur zulässig sein, wenn Fließkomma-Parameter 0
  und Fließkomma-Parameter 1 unterschiedliches Vorzeichen haben!}
  if D^.E[0] * D^.E[1] < 0 then
    CanSimulateDLL := 1
  else
    CanSimulateDLL := 0;
end;
```

Beispiele für die Realisierung von CanSimulateDLL

CanSimulateDLL2

Signatur

```
function CanSimulateDLL2(T, DeltaT:Extended;
                         D:PParameterStruct):Integer;
                                   export stdcall;

int _export _stdcall CanSimulateDLL(long double T,
                                    long double DeltaT,
                                    PParameterStruct D)
```

Parameter *T* ist die eingestellte Simulationsdauer

DeltaT ist die eingestellte Simulationsschrittweite

D ist ein Zeiger auf die Struktur *TParameterStruct*.

Rückgabewert Der Rückgabewert '0' zeigt an, dass nicht simuliert werden kann, der Rückgabewert von '1', dass dieser Block simulationsbereit ist.

Funktion Kann alternativ zu *CanSimulateDLL* benutzt werden, wenn innerhalb der Funktion die Simulationsdauer oder -schrittweite benötigt wird.

InitSimulationDLL

Signatur

```
procedure InitSimulationDLL (D:PParameterStruct;
                             Inputs:PInputArray;
                             Outputs:POutputArray);
                             export stdcall;

void _export _stdcall InitSimulationDLL (PParameterStruct D,
                              TInputArray FAR Inputs,
                              TOutputArray FAR Outputs)
```


Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Inputs ist ein Zeiger auf die Struktur *TInputArray.*

Outputs ist ein Zeiger auf die Struktur *TOutputArray.*

Funktion *InitSimulationDLL* initialisiert den Block für den Zeitpunkt $t = 0$ bzw. den ersten Simulationsschritt. Hier können also Voreinstellungen vorgenommen werden, die vor Beginn der eigentlichen Simulationsschleife notwendig werden (z. B. Setzen von Anfangswerten oder inneren Zustandsgrößen, Vorbelegung der Ausgangswerte usw.). Sofern erforderlich, können innerhalb dieser Funktion z. B. auch Daten, die für die Simulation benötigt werden, aus einer externen Parameterdatei gelesen werden (siehe Beispiel *Kennfeld-DLL* an späterer Stelle). Die aktuellen Eingangswerte des Blocks werden über den Zeiger *Inputs* übergeben; die berechneten Ausgangswerte müssen über den Zeiger *Outputs* zurückgeliefert werden.

Beispiel:

Nachfolgendes Listing zeigt eine beispielhafte Realisierung von *InitSimulationDLL* für einen Block mit zwei Eingängen und zwei Ausgängen. In diesem Fall soll zum Zeitpunkt $t = 0$ der erste Eingang auf den ersten Ausgang durchgeschaltet werden und der zweite Ausgang zu 0 initialisiert werden.

```
procedure InitSimulationDLL (D:PParameterStruct; Inputs:PInputArray;
                             Outputs:POutputArray); export stdcall;
begin
  Outputs^[1] := Inputs^[1];  {Ausgang 1 auf Eingang 1}
  Outputs^[2] := 0.0;         {Ausgang 2 zu 0 initialisieren}
end;
```

Beispiel für die Realisierung von InitSimulationDLL

InitSimulationDLL2

Signatur

```
procedure InitSimulationDLL2(DeltaT:Extended;
                                  D:PParameterStruct;
                                  Inputs:PInputArray;
                                  Outputs:POutputArray);
                                  export stdcall;

void _export _stdcall InitSimulationDLL2
                                  (long double DeltaT,
                                   PParameterStruct D,
                                   TInputArray FAR Inputs,
                                   TOutputArray FAR Outputs)
```

Parameter *DeltaT* ist die Simulationsschrittweite.

D ist ein Zeiger auf die Struktur *TParameterStruct.*

Inputs ist ein Zeiger auf die Struktur *TInputArray.*

Outputs ist ein Zeiger auf die Struktur *TOutputArray*.

Funktion *InitSimulationDLL2* kann alternativ zu *InitSimulationDLL* benutzt werden, wenn im Initialisierungsschritt die Abtastzeit *DeltaT* benötigt wird.

SimulateDLL

Signatur

```
Procedure SimulateDLL(T:Extended; D:PParameterStruct;
                      Inputs:PInputArray;
                      Outputs:POutputArray);
                      export stdcall;

void _export _stdcall SimulateDLL(long double T,
                            PParameterStruct D,
                            TInputArray FAR Inputs,
                            TOutputArray FAR Outputs)
```

Parameter *T* ist der Zeitwert des augenblicklichen Simulationsschrittes.

D ist ein Zeiger auf die Struktur *TParameterStruct.*

Inputs ist ein Zeiger auf die Struktur *TInputArray.*

Outputs ist ein Zeiger auf die Struktur *TOutputArray*.

Funktion *SimulateDLL* enthält den eigentlichen numerischen Kern des User-Blocks, d. h. den zu jedem Simulationsschritt abzuarbeitenden Algorithmus. Diese Funktion wird zu jedem Simulationsschritt aufgerufen. Der Parameter *T* enthält dabei den momentanen Zeitwert, der Parameter *D* einen Zeiger auf die Blockparameter. Die aktuellen Eingangswerte des Blocks werden über den Zeiger *Inputs* übergeben; die berechneten Ausgangswerte müssen über den Zeiger *Outputs* zurückgeliefert werden.

Wird im Simulationsalgorithmus auch die Simulationsschrittweite ΔT benötigt, kann statt der Funktion *SimulateDLL* die Funktion *SimulateDLL2* (wird nachfolgend beschrieben) benutzt werden, der die Schrittweite als zusätzlicher Parameter übergeben wird.

Beispiel:

Folgendes Listing zeigt die Realisierung von *SimulateDLL* für einen Block mit zwei Eingängen und zwei Ausgängen. Am ersten Ausgang wird der arithmetische Mittelwert beider Eingänge - zusätzlich multipliziert mit Fließkomma-Parameter 0 - und am zweiten Ausgang der geometrische Mittelwert - multipliziert mit Fließkomma-Parameter 1 - ausgegeben.


```
Procedure SimulateDLL(T:Extended; D:PParameterStruct;
                      Inputs:PInputArray;
                      Outputs:POutputArray); export stdcall;
begin
  Outputs^[1] := D^.E[0] * (Inputs^[1] + Inputs^[2]) | 2;
  Outputs^[2] := D^.E[1] * sqrt((Inputs^[1] * Inputs^[2]));
end;
```

Beispiel für die Realisierung von SimulateDLL

SimulateDLL2

Signatur

```
Procedure SimulateDLL2(T, DeltaT:Extended;
                       D:PParameterStruct;
                       Inputs:PInputArray;
                       Outputs:POutputArray);
                       export stdcall;
```

```
void _export _stdcall SimulateDLL2(long double T,
                                   long double DeltaT,
                                   PParameterStruct D,
                                   TInputArray FAR Inputs,
                                   TOutputArray FAR Outputs)
```

Parameter *T* ist der Zeitwert des augenblicklichen Simulationsschrittes

DeltaT ist die Simulationsschrittweite.

D ist ein Zeiger auf die Struktur *TParameterStruct.*

Inputs ist ein Zeiger auf die Struktur *TInputArray*.

Outputs ist ein Zeiger auf die Struktur *TOutputArray.*

Funktion *SimulateDLL2* kann statt der Funktion *SimulateDLL* benutzt werden, wenn zur Verarbeitung auch die Simulationsschrittweite ΔT benötigt wird.

SimulateDLL3

Signatur

```
Function SimulateDLL3(T, DeltaT:Extended;
                      D:PParameterStruct;
                      Inputs:PInputArray;
                      Outputs:POutputArray): integer;
                       export stdcall;
```

```
int _export _stdcall SimulateDLL3(long double T,
                                  long double DeltaT,
                                  PParameterStruct D,
                                  TInputArray FAR Inputs,
                                  TOutputArray FAR Outputs)
```

Parameter *T* ist der Zeitwert des augenblicklichen Simulationsschrittes

DeltaT ist die Simulationsschrittweite.

D ist ein Zeiger auf die Struktur *TParameterStruct.*

Inputs ist ein Zeiger auf die Struktur *TInputArray.*

Outputs ist ein Zeiger auf die Struktur *TOutputArray*.

Rückgabewert Ein Rückgabewert von 0 zeigt an, dass die Simulation fortgeführt werden soll; jeder andere Rückgabewert führt zu einem Simulationsabbruch nach dem aktuellen Simulationsschritt.

Funktion *SimulateDLL3* kann statt der Funktion *SimulateDLL* oder *SimulateDLL2* benutzt werden, wenn die Simulation von der User-DLL unter bestimmten Bedingungen abgebrochen werden soll.

EndSimulationDLL

Signatur

```
procedure EndSimulationDLL; export stdcall;
void _export _stdcall EndSimulationDLL(void)
```

Funktion *EndSimulationDLL* wird aufgerufen, sobald die Simulation beendet wurde. In dieser Funktion können somit z. B. geöffnete Dateien geschlossen werden. In den meisten Anwendungen kann diese Funktion jedoch leer bleiben.

Statt *EndSimulationDLL* kann auch die Funktion *EndSimulationDLL2* (wird nachfolgend beschrieben) benutzt werden, der als zusätzlicher Parameter ein Zeiger auf die Struktur *TParameterStruct* übergeben wird.

Beispiel:

Nachfolgendes Listing zeigt ein Beispiel für die Anwendung von *EndSimulationDLL*.

```
procedure EndSimulationDLL; export stdcall;
begin
  Close(DataFile);   {Parameterdatei schließen}
end;
```

Beispiel für die Anwendung von EndSimulationDLL

EndSimulationDLL2

Signatur

```
procedure EndSimulationDLL2(D: PParameterStruct);
                                        export stdcall;
```



```
void _export _stdcall EndSimulationDLL2
                                    (PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion *EndSimulationDLL2* kann statt der Funktion *EndSimulationDLL* benutzt werden, wenn zur Verarbeitung auch die Parameterstruktur *TParameterStruct* benötigt wird.

SetEnhancedInformation

Signatur

```
procedure SetEnhancedInformation
                                (DeltaT, TSimu: extended;
                                 D: PParameterStruct);
                                 export stdcall;

void _export _stdcall SetEnhancedInformation
                                  (long double DeltaT,
                                   long double TSimu,
                                   PParameterStruct D)
```

Parameter *DeltaT* ist die Simulationsschrittweite.

TSimu ist die Simulationsdauer.

D ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion *SetEnhancedInformation* wird nur in seltenen Fällen benötigt. Die Funktion liefert dem User-Block die eingestellte Simulationsschrittweite und -dauer zur weiteren Verwendung. Beide Größen können z. B. in nicht benutzten Fließkomma-Parametern zur weiteren Nutzung abgelegt werden. Die Funktion wird von BORIS zu Beginn eines jeden Simulationslaufes (d. h. vor *InitSimulationDLL*) aufgerufen.

Beispiel:

In nachfolgendem Beispiel werden *DeltaT* und *TSimu* in den Fließkomma-Parametern 3 und 4 gespeichert.

```
procedure SetEnhancedInformation(DeltaT, TSimu: extended;
                                 D: PParameterStruct); export stdcall;
begin
  D^.E[3] := DeltaT;
  D^.E[4] := TSimu;
end;
```

Beispiel für die Anwendung von SetEnhancedInformation

SetEnhancedInformation2

Signatur

```
procedure SetEnhancedInformation2
                                (DeltaT, TSimu: extended;
                                 D1: PConnectedArray;
                                 D2: PParameterStruct);
                                 export stdcall;

void _export _stdcall SetEnhancedInformation2
                                  (long double DeltaT,
                                   long double TSimu,
                                   PConnectedArray D1,
                                   PParameterStruct D2)
```

Parameter *DeltaT* ist die Simulationsschrittweite.

TSimu ist die Simulationsdauer.

D1 ist ein Zeiger auf die Struktur *TConnectedArray.*

D2 ist ein Zeiger auf die Struktur *TParameterStruct.*

Funktion Diese Funktion kann an Stelle von *SetEnhancedInformation* benutzt werden, wenn zusätzlich Informationen darüber benötigt werden, welcher Blockeingang tatsächlich eine Verbindung aufweist. Die Funktion wird von BORIS zu Beginn eines jeden Simulationslaufes (d. h. vor *InitSimulationDLL*) aufgerufen.

HasDelayDLL

Signatur

```
function HasDelayDLL(D: PParameterStruct): integer;
                                           export stdcall;

int _export _stdcall HasDelayDLL(PParameterStruct D)
```

Parameter *D* ist ein Zeiger auf die Struktur *TParameterStruct.*

Rückgabewert Ein Rückgabewert ungleich 0 kennzeichnet, dass der Block eine Verzögerung aufweist.

Funktion Diese Funktion kann benutzt werden, um zu kennzeichnen, dass die User-DLL eine Verzögerung aufweist und somit zum Auflösen algebraischer Schleifen benutzt werden kann. Dies hat zur Konsequenz, dass von BORIS beim Aufruf der Funktion *SimulateDLL* bzw. *SimulateDLL2* oder *SimulateDLL3* im Vektor *Inputs* nicht die aktuellen Eingangswerte des Blocks, sondern diejenigen vom vorangegangenen Simulationsschritt übergeben werden; beim Aufruf von *InitSimulationDLL* bzw. *InitSimulationDLL2* werden entsprechend *gar keine gültigen Werte* in *Inputs* übergeben. Die Ausgangswerte des Blocks müssen demzufolge in die-

sem Fall bei der Initialisierung des Blocks auf feste, d. h. eingangsgrößenunabhängige Werte gesetzt werden.

Wird die Funktion nicht benutzt, wird die User-DLL als Block ohne Verzögerung verwendet.

Beispiele

Beispiel 1: Eine simple User-DLL

Wir wollen zunächst beginnen mit einem einfachen Beispiel einer User-DLL mit zwei Eingängen und einem Ausgang. Der Block soll in Abhängigkeit vom ersten Eingang (Steuereingang) x_1 den Wert des zweiten Eingangs (Dateneingang) x_2 mit unterschiedlichen Verstärkungsfaktoren k_1, k_2, k_3 multiplizieren und am Ausgang *y* ausgeben. Über einen Schalter-Parameter b_1 soll der Ausgang zusätzlich invertiert werden können:

- Falls x_1 negativ ist, soll x_2 mit k_1 multipliziert werden.
- Falls x_1 null ist, soll x_2 mit k_2 multipliziert werden.
- Falls x_1 positiv ist, soll x_2 mit k_3 multipliziert werden.
- Falls b_1 gesetzt ist, soll der Ausgang zusätzlich invertiert werden.

Nachfolgendes Listing zeigt die Realisierung der DLL in Pascal. Die fertig compilierte DLL befindet sich als DLLDEMO1.DLL im *UserDLLs*-Verzeichnis, der zugehörige Quelltext unter DLLDEMO1.DPR im Unterverzeichnis *Sources*. Alle wesentlichen Kommentare zur DLL sind im Quelltext vermerkt.

```
Library DLLDemo1;
(************************************************)
(*                                              *)
(*       Einfache Demo-DLL für BORIS            *)
(*       (Beispiel 1 des Handbuchs)             *)
(*                                              *)
(************************************************)

uses WinProcs,
  SysUtils,
  WinTypes;

type
  PParameterStruct=^TParameterStruct;
  TParameterStruct=packed record
    NuE : Byte;                    {Anzahl reeller Zahlenwerte}
    NuI : Byte;                    {Anzahl ganzer Zahlenwerte}
```

```
    NuB : Byte;                     {Anzahl Schalter}
    E:Array[0..31] of Extended;     {reelle Zahlenwerte}
    I:Array[0..31] of Integer;      {ganze Zahlenwerte}
    B:Array[0..31] of Byte;         {Schalter}
    D:Array[0..255] of char;        {event. Dateiname für weitere Daten.}
    EMin:Array[0..31] of Extended;  {untere Eingabegrenze}
    EMax:Array[0..31] of Extended;  {obere Eingabegrenze}
    IMin:Array[0..31] of Integer;   {untere Eingabegrenze}
    IMax:Array[0..31] of Integer;   {obere Eingabegrenze}
    NaE : Array[0..31,0..40] of char;    {Namen der reellen Zahlenwerte}
    NaI : Array[0..31,0..40] of char;    {Namen der ganzen Zahlenwerte}
    NaB : Array[0..31,0..40] of char;    {Namen der Schalter}
    {den Rest der Variablen von TParameterStruct können wir uns sparen,
     da wir ihn nicht benötigen!}
  end;

  PDialogEnableStruct=^TDialogEnableStruct;
  TDialogEnablestruct=packed record
    AllowE: Longint;                { Soll die Eingabe eines Wertes   }
    AllowI: Longint;                { un-/zulässig sein so ist das Bit}
    AllowB: Longint;                { des Allow?-Feldes 0 bzw. 1}
    AllowD: Byte;
  end;

  PNumberOfInputsOutputs=^TNumberOfInputsOutputs;
  TNumberOfInputsOutputs=packed record
    Inputs :Byte;                   {Anzahl Eingänge}
    Outputs:Byte;                   {Anzahl Ausgänge}
    NameI : Array[0..49,0..40] of char;
    NameO : Array[0..49,0..40] of char;
  end;

  PInputArray = ^TInputArray;
  TInputArray = packed array[1..50] of extended;
  POutputArray = ^TOutputArray;
  TOutputArray = packed array[1..50] of extended;


procedure GetParameterStruct(D:PParameterStruct);export stdcall;
begin
  D^.NuE:=3; {3 Fließkomma-Parameter}
  D^.NuI:=0; {0 Integer-Parameter}
  D^.NuB:=1; {1 Schalter-Parameter}
  {Namen der Parameter}
  StrCopy(D^.NaE[0], 'Verstärkung k1 für Steuereing. < 0');
  StrCopy(D^.NaE[1], 'Verstärkung k2 für Steuereing. = 0');
  StrCopy(D^.NaE[2], 'Verstärkung k3 für Steuereing. > 0');
  StrCopy(D^.NaB[0], 'Ausgang invertiert');
  {Parameter initialisieren}
  D^.E[0] := 1;
  D^.E[1] := 5;
  D^.E[2] := 10;
  D^.B[0] := 0;
  {Grenzwerte für Parameter festlegen}
  D^.EMin[0] := 0.0; D^.EMax[0] := 100000;
  D^.EMin[1] := 0.0; D^.EMax[1] := 100000;
  D^.EMin[2] := 0.0; D^.EMax[2] := 100000;
end;

procedure GetDialogEnableStruct(D:PDialogEnableStruct;
                                D2:PParameterStruct); export stdcall;
begin
```



```
  {Alle Dialogelemente sollen jederzeit zugänglich sein}
  D^.AllowE := $FFFFFFFF;
  D^.AllowI := $FFFFFFFF;
  D^.AllowB := $FFFFFFFF;
end;

procedure GetNumberOfInputsOutputs(D:PNumberOfInputsOutputs);
                                                    export stdcall;
begin
  D^.Inputs:=2;            { Der Block soll zwei Eingänge und}
  D^.Outputs:=1;           {            einen Ausgang haben !}
  StrPCopy(D^.NameI[0],'Dateneingang');    {Namensgebung des 1. Eingangs}
  StrPCopy(D^.NameI[1],'Steuereingang');   {Namensgebung des 2. Eingangs}
  StrPCopy(D^.NameO[0],'Ausgang');         {Namensgebung des 1. Ausgangs}
end;

function CanSimulateDLL(D:PParameterStruct):Integer; export stdcall;
begin
  {Simulation immer zulässig!}
  CanSimulateDLL := 1;
end;

procedure SimulateDLL(T:Extended;D:PParameterStruct;Inputs:PInputArray;
                      Outputs:POutputArray);export stdcall;
begin
  if Inputs^[1] < 0 then {Eingang 1 negativ}
    Outputs^[1] := D^.E[0] * Inputs^[2]
  else if Inputs^[1] = 0 then {Eingang 1 null}
    Outputs^[1] := D^.E[1] * Inputs^[2]
  else {Eingang 1 positiv}
    Outputs^[1] := D^.E[2] * Inputs^[2];
  if D^.B[0] = 1 then {Schalter-Parameter 0 = 1 ==> Ausgang invertieren}
    Outputs^[1] := -Outputs^[1];
end;

procedure InitSimulationDLL(D:PParameterStruct;Inputs:PInputArray;
                            Outputs:POutputArray);export stdcall;
begin
  {Der Initialisierungsschritt soll genauso durchgeführt werden wie alle
   Simulationsschritte. Also rufen wir einfach SimulateDLL auf!}
  SimulateDLL(0, D, Inputs, Outputs);
end;

procedure EndSimulationDLL; export stdcall;
begin
  {wird nicht benötigt}
end;

function SetInputChar: PChar; export stdcall;
begin
  {Steuereingang mit 'S', Dateneingang mit 'D' beschriften}
  SetInputChar := 'SD';
end;

procedure IsUserDLL32; export stdcall;
begin
end;

{Exportieren der notwendigen Funktionen und Prozeduren }
exports
  GetParameterStruct,
  GetDialogEnableStruct,
```



```
  GetNumberOfInputsOutputs,
  CanSimulateDLL,
  InitSimulationDLL,
  SimulateDLL,
  EndSimulationDLL,
  SetInputChar,
  IsUserDLL32;
begin
  {Initialisierung der DLL (nicht notwendig)}
end.
```

Pascal-Listing zu Beispiel 1

Beispiel 2: Benutzerdefiniertes Kennfeld

Als Erweiterung zur benutzerdefinierten Kennlinie, die in der BORIS-Systemblockbibliothek ja standardmäßig verfügbar ist, soll eine User-DLL zur Realisierung eines benutzerdefinierten *Kennfelds* der Form $z = f(x, y)$ erstellt werden. Die User-DLL soll folgende Spezifikationen erfüllen:

- Das Kennfeld soll in Matrixform als FWM-Datei (siehe Abschnitt *WinFACT-Dateitypen* in Kapitel *2 Grundlagen*) eingelesen werden. Der Name der Datei soll über den Parameterdialog vorgebbar sein. Der zugrundeliegende Abszissenbereich $[\, x_{\min}, x_{\max}\,]$ bzw. $[\, y_{\min}, y_{\max}\,]$ soll ebenfalls über den Parameterdialog des User-Blocks einstellbar sein. Die Matrix soll maximal die Dimension 20×20 aufweisen können.
- Zwischen den Stützpunkten soll auf Wunsch linear interpoliert werden können. Liegt ein Eingangswertepaar (x, y) außerhalb des von der Stützstellenmatrix erfassten Bereichs, soll extrapoliert werden.
- Neben dem eigentlichen *z*-Ausgang soll ein zweiter Ausgang anzeigen, ob ein Eingangswertepaar (x, y) außerhalb des von der Stützstellenmatrix erfassten Bereichs liegt, also extrapoliert wird. In diesem Fall soll dieser Ausgang High-Pegel, sonst Low-Pegel aufweisen. Die Werte für Low- und High-Pegel sollen ebenfalls über den Parameterdialog einstellbar sein.

Nachfolgendes Listing zeigt die Realisierung in Pascal. Die fertig compilierte DLL befindet sich als DRDFELD.DLL im *UserDLLs*-Verzeichnis, der zugehörige Quelltext unter DRDFELD.DPR im Unterverzeichnis *Sources*. Alle wesentlichen Kommentare zur DLL sind im Quelltext vermerkt.

```
Library DRDFeld;

(*************************************************)
(*                                               *)
(*          Kennfeld-User-DLL für BORIS          *)
(*           (Beispiel 2 des Handbuchs)          *)
```



```
(*                                             *)
(***********************************************)

uses Windows, SysUtils;

type
  PParameterStruct=^TParameterStruct;
  TParameterStruct= packed record
    NuE : Byte;                    {Anzahl reeller Zahlenwerte}
    NuI : Byte;                    {Anzahl ganzer Zahlenwerte}
    NuB : Byte;                    {Anzahl Schalter}
    E:Array[0..31] of Extended;    {reelle Zahlenwerte}
    I:Array[0..31] of Integer;     {ganze Zahlenwerte}
    B:Array[0..31] of Byte;        {Schalter}
    D:Array[0..255] of char;       {event. Dateiname für weitere Daten.}
    EMin:Array[0..31] of Extended;
    EMax:Array[0..31] of Extended;
    IMin:Array[0..31] of Integer;
    IMax:Array[0..31] of Integer;
    NaE : Array[0..31,0..40] of char;
    NaI : Array[0..31,0..40] of char;
    NaB : Array[0..31,0..40] of char;
    UserDataPtr: Pointer;
    {Den Rest von TParameterStruct können wir uns sparen, da wir ihn
     nicht benötigen!}
  end;

  PDialogEnableStruct=^TDialogEnableStruct;
  TDialogEnablestruct=packed record
    AllowE: Longint;               { Soll die Eingabe eines Wertes   }
    AllowI: Longint;               { un-/zulässig sein so ist das Bit}
    AllowB: Longint;               { des Allow?-Feldes 0 bzw. 1}
    AllowD: Byte;
  end;

  PNumberOfInputsOutputs=^TNumberOfInputsOutputs;
  TNumberOfInputsOutputs=packed record
    Inputs :Byte;                        {Anzahl Eingänge}
    Outputs:Byte;                        {Anzahl Ausgänge}
    NameI : Array[0..49,0..40] of char;
    NameO : Array[0..49,0..40] of char;
  end;

  PInputArray = ^TInputArray;
  TInputArray = packed array[1..50] of extended;
  POutputArray = ^TOutputArray;
  TOutputArray = packed array[1..50] of extended;


  {TUserData enthält die Kennfelddaten}
  TSingleMatrix = Array[1..20, 1..20] of Single;
  PUserData = ^TUserData;
  TUserData = record
                z: TSingleMatrix; {Funktionswertmatrix}
                nx,ny: Word;      {Anzahl Spalten bzw. Zeilen der Matrix}
                dx,dy: Extended;  {x- bzw. y-Abstand zwischen zwei
                                   Stützpunkten}
              end;

Const
  {Zwei Konstanten-Deklarationen für später...}
  SingleMin=-3.4E38;
```

```
  SingleMax= 3.4E38;


procedure GetParameterStruct(D:PParameterStruct);export stdcall;
begin
  D^.NuE:=6;              {Sechs Fließpunktparameter}
  D^.NuI:=0;              {Keine Ganzzahlparameter }
  D^.NuB:=1;              {Ein Schalter}
  StrPCopy(D^.D,'*.fwm');  {Dateien mit Endung fwm zulassen}
  {Initialisierung der verwendeten Daten}
  D^.B[0]:=0;
  D^.E[0]:=-1; D^.EMin[0]:=SingleMin; D^.EMax[0]:=SingleMax;
  D^.E[1]:=1;  D^.EMin[1]:=SingleMin; D^.EMax[1]:=SingleMax;
  D^.E[2]:=-1; D^.EMin[2]:=SingleMin; D^.EMax[2]:=SingleMax;
  D^.E[3]:=1;  D^.EMin[3]:=SingleMin; D^.EMax[3]:=SingleMax;
  D^.E[4]:=0;  D^.EMin[4]:=SingleMin; D^.EMax[4]:=SingleMax;
  D^.E[5]:=5;  D^.EMin[5]:=SingleMin; D^.EMax[5]:=SingleMax;
  {Namensgebung max. 40 Zeichen !}
  StrPCopy(D^.NaE[0],'x - Achsenanfang');
  StrPCopy(D^.NaE[1],'x - Achsenende');
  StrPCopy(D^.NaE[2],'y - Achsenanfang');
  StrPCopy(D^.NaE[3],'y - Achsenende');
  StrPCopy(D^.NaE[4],'Low-Pegel für Ausgang 2');
  StrPCopy(D^.NaE[5],'High-Pegel für Ausgang 2');
  StrPCopy(D^.NaB[0],'lineare Interpolation');
end;

procedure GetDialogEnableStruct(D:PDialogEnableStruct;
                                D2:PParameterStruct); export stdcall;
begin
  {Alle Dialogelemente sollen jederzeit zugänglich sein!}
  D^.AllowE:=$FFFFFFFF;
  D^.AllowB:=$FFFFFFFF;
  D^.AllowI:=$FFFFFFFF;
  D^.AllowD:= 1;
end;

procedure GetNumberOfInputsOutputs(D:PNumberOfInputsOutputs);export stdcall;
begin
  D^.Inputs:=2;            { Der Block soll zwei Eingänge und}
  D^.Outputs:=2;           {               zwei Ausgänge haben !}
  StrPCopy(D^.NameI[0],'Abszissen-Wert (x)');    {Name des 1. Eingangs}
  StrPCopy(D^.NameI[1],'Abszissen-Wert (y)');    {Name des 2. Eingangs}
  StrPCopy(D^.NameO[0],'Ordinate (z) ');         {Name des 1. Ausgangs}
  StrPCopy(D^.NameO[1],'Extrapolation ON/OFF');  {Name des 2. Ausgangs}
end;

function CanSimulateDLL(D:PParameterStruct):Integer; export stdcall;
var Datei: Text;
    Dateiname : Array[0..255] of char;
begin
  {Simulation nur zulassen, falls die angegebene FWM-Datei auch existiert!}
  StrCopy(Dateiname, D^.D);
  if Pos('*',StrPas(Dateiname))=0 then begin
    Assign(Datei,Dateiname);
    {$I-} Reset(Datei); {$I+}
    if IOResult = 0 then begin
      CanSimulateDLL := 1;
      Close(Datei);
    end else CanSimulateDLL := 0;
  end else CanSimulateDLL:=0;
end;
```



```
procedure SimulateDLL(T:Extended;D:PParameterStruct;
                  Inputs:PInputArray;Outputs:POutputArray);export stdcall;
var ix, iy: integer;
    x0, y0, xp, yp: Extended;
    Extrapolation, Interpolation: boolean;
begin
  with PUserData(D^.UserDataPtr)^ do begin
    Extrapolation := (Inputs^[1]<D^.E[0]) or (Inputs^[1]>D^.E[1]) or
                     (Inputs^[2]<D^.E[2]) or (Inputs^[2]>D^.E[3]);
    if Extrapolation then
      Outputs^[2]:=D^.E[5]   {Ausgang 2 auf High-Pegel}
    else
      Outputs^[2]:=D^.E[4];  {Ausgang 2 auf Low-Pegel}
    Interpolation := D^.B[0] = 1;
    {Indizes ix, iy des nächstgelegenen Stützpunkts bestimmen}
    ix := trunc((Inputs^[1] - D^.E[0])/dx) + 1;
    iy := trunc((Inputs^[2] - D^.E[2])/dy) + 1;
    if ix < 1 then ix := 1;
    if iy < 1 then iy := 1;
    if Interpolation then begin
      if ix > nx-1 then ix := nx-1;
      if iy > ny-1 then iy := ny-1;
      x0 := D^.E[0] + (ix-1)*dx; {Referenzpunkt, x-Koordinate}
      y0 := D^.E[2] + (iy-1)*dy; {Referenzpunkt, y-Koordinate}
      xp := (z[ix+1, iy] - z[ix, iy]) / dx; {Steigung in x-Richtung}
      yp := (z[ix, iy+1] - z[ix, iy]) / dy; {Steigung in y-Richtung}
      Outputs^[1] := z[ix, iy] + (Inputs^[1] - x0)*xp + (Inputs^[2]-y0)*yp;
    end else begin
      if ix > nx then ix := nx;
      if iy > ny then iy := ny;
      Outputs^[1] := z[ix, iy];
    end;
  end;
end;

procedure InitSimulationDLL(D:PParameterStruct;Inputs:PInputArray;
                            Outputs:POutputArray);export stdcall;
var i, j: Integer;
    Datei: text;
begin
  {Erst Parameterdatei einlesen...}
  assign(Datei, D^.D);
  reset(Datei);
  with PUserData(D^.UserDataPtr)^ do begin
    readln(Datei,ny,nx);
    for i:=1 to ny do
      for j:=1 to nx do readln(Datei,z[j, i]);
    dx:=(D^.E[1]-D^.E[0])/(nx-1); {Segmentgröße in x-Richtung errechnen!}
    dy:=(D^.E[3]-D^.E[2])/(ny-1); {Segmentgröße in y-Richtung errechnen!}
  end;
  close(Datei);
  {...und dann einen normalen Simulationsschritt anschließen}
  SimulateDLL(0, D, Inputs, Outputs);
end;

procedure EndSimulationDLL;export stdcall;
begin
  {wird nicht benötigt}
end;

procedure InitUserData(D: PParameterStruct); export stdcall;
```



```
begin
  {Speicherplatz für User-Daten anlegen}
  GetMem(D^.UserDataPtr, sizeof(TUserData));
end;

procedure DisposeUserData(D: PParameterStruct); export stdcall;
begin
  {Speicherplatz für User-Daten wieder freigeben}
  FreeMem(D^.UserDataPtr, sizeof(TUserData));
end;

function SetInputChar: PChar; export stdcall;
begin
  {Ersten Eingang mit 'x', zweiten mit 'y' beschriften}
  SetInputChar := 'xy';
end;

function SetOutputChar: PChar; export stdcall;
begin
  {Ersten Ausgang mit 'z', zweiten mit 'E' beschriften}
  SetOutputChar := 'zE';
end;

procedure IsUserDLL32; export stdcall;
begin
end;

{Exportieren der notwendigen Funktionen und Prozeduren }
exports
  GetParameterStruct, GetDialogEnableStruct, GetNumberOfInputsOutputs,
  CanSimulateDLL, InitSimulationDLL, SimulateDLL,
  InitUserData, DisposeUserData, EndSimulationDLL,
  SetInputChar, SetOutputChar, IsUserDLL32;

begin
  {Weitere Initialisierung der DLL (nicht notwendig).}
end.
```

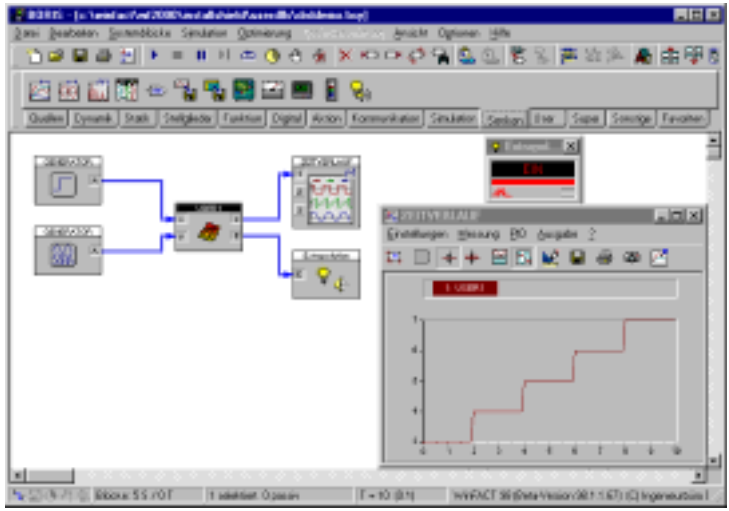
Pascal-Listing für Beispiel 2

Anwendungsbeispiel für Kennfeld-DLL

Es soll ein Kennfeld definiert werden für den Bereich $0 \le x \le 4,\ 0 \le y \le 4$, bei dem der Ausgangswert (z-Achse) linear von 1 bis 9 ansteigt. Das Kennfeld soll in beiden Richtungen über jeweils 5 Stützpunkte festgelegt sein. Dann hat die zugehörige Kennfeldmatrix folgendes Aussehen:

$$\underline{M} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{pmatrix}.$$

Die entsprechende FWM-Datei befindet sich unter dem Namen DRDDEMO.FWM im Verzeichnis *UserDLLs*. Es soll der Verlauf der Ausgangsgröße für den Fall simuliert werden, dass die Eingangsgröße *x* konstant 2 ist und die Eingangsgröße *y* linear von 0 auf 5 ansteigt. Es soll dabei keine Interpolation stattfinden. Nachfolgende Grafik zeigt das entsprechende Simulationsergebnis. Die zugehörige Systemdatei befindet sich unter dem Namen DRDDEMO.BSY ebenfalls im Verzeichnis *UserDLLs*.



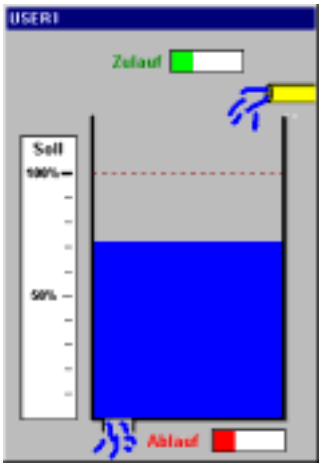
Simulationsergebnis

Beispiel 3: Visualisierung einer Füllstandsregelung

Es soll eine User-DLL erstellt werden, die zur Visualisierung einer Füllstandsregelung dient. Dabei soll beim Starten des ersten Simulationslaufs ein Ausgabefenster erscheinen, in dem die Eingangsgrößen des Blocks (Zulauf, Ablauf, Füllhöhe) visualisiert werden (siehe nachfolgende Grafik). Dieses Ausgabefenster soll erst dann wieder verschwinden, wenn der Block gelöscht wird. Bei der Programmierung soll davon ausgegangen werden, dass alle Eingangsgrößen zwischen 0 und 1 liegen (entsprechend 0 bzw. 100% Zulauf, Ablauf oder Füllhöhe).

Nachfolgendes Listing zeigt die Realisierung der DLL allein auf Basis der Windows-API. Die fertige DLL befindet sich unter dem Namen FUELLSTD.DLL im Verzeichnis *UserDLLs*, der Quelltext FUELLSTD.DPR im Unterverzeichnis *Sources*. Eine wesentlich einfachere Programmierung derartiger Blocktypen lässt sich mit Entwicklungsumgebungen wie DELPHI,

VISUAL BASIC etc. erreichen; eine ganze Reihe von in DELPHI 3 realisierten Beispielen (teilweise mit Quelltexten) finden Sie im *UserDLLs*- bzw. *Sources*-Verzeichnis.



Visualisierungsfenster der User-DLL

```
Library FuellStd;

(*********************************************)
(*                                           *)
(*        DLL zur Visualisierung einer       *)
(*            Füllstandsregelung             *)
(*         (Beispiel 3 des Handbuchs)        *)
(*                                           *)
(*********************************************)

uses WinProcs, WinTypes, Windows, SysUtils, Messages;

{$R FUELLBMP}  {RES-Datei Hintergrund-Bitmap für das Ausgabefenster}


const
  cxBitmap = 250;  (* Breite des Bitmaps *)
  cyBitmap = 350;  (* Höhe des Bitmaps   *)

 {Damit ggfls. auch mehrere DLL-Blöcke dieses Typs gleichzeitig simuliert
  werden können, müssen wir uns für jedes Fenster das Fensterhandle und die
  aktuellen Eingangswerte merken. Außerdem soll jedes Fenster den Namen des
  zugehörigen User-DLL-Blocks im Titel anzeigen. Eleganterweise macht man
  das in einer verketteten Liste. Damit es nicht ganz so kompliziert wird,
  wählen wir hier aber ein statisches Array der Dimension 100 (mehr Blöcke
  wird wohl kaum jemand nehmen!)}
var
  nWindows: integer;  {enthält die Anzahl der geöffneten Fenster}
  LevelWindows: array[1..100] of record
    HW: HWnd;                    {Fensterhandle}
    VPos, VNeg, Height: extended; {aktuelle Eingangswerte des Blocks:
```

```
                                    Zulauf, Ablauf, Füllstand}
  end;

type
  PParameterStruct=^TParameterStruct;
  TParameterStruct=packed record
    NuE : Byte;                       {Anzahl reeller Zahlenwerte}
    NuI : Byte;                       {Anzahl ganzer Zahlenwerte}
    NuB : Byte;                       {Anzahl Schalter}
    E:Array[0..31] of Extended;       {reelle Zahlenwerte}
    I:Array[0..31] of Integer;        {ganze Zahlenwerte}
    B:Array[0..31] of Byte;           {Schalter}
    D:Array[0..255] of char;          {event. Dateiname für weitere Daten.}
    EMin:Array[0..31] of Extended;
    EMax:Array[0..31] of Extended;
    IMin:Array[0..31] of Integer;
    IMax:Array[0..31] of Integer;
    NaE : Array[0..31,0..40] of char; {Namen der rellen Zahlenwerte}
    NaI : Array[0..31,0..40] of char; {Namen der ganzen Zahlenwerte}
    NaB : Array[0..31,0..40] of char; {Namen der Schalter}
    UserDataPtr: Pointer;             {benutzerdef. Daten (nicht benötigt)}
    ParentPtr: Pointer;               {Zeiger auf User-DLL-Block}
    ParentHWnd: HWnd;                 {BORIS-Fensterhandle}
    ParentName: PChar;                {Name des User-DLL-Blocks}
    UserHWindow: HWnd;                {Handle des Ausgabefensters}
    DataFile: text;                   {Text-Datei (hier nicht benötigt)}
  end;

  PDialogEnableStruct=^TDialogEnableStruct;
  TDialogEnablestruct=packed record
    AllowE: Longint;                { Soll die Eingabe eines Wertes   }
    AllowI: Longint;                { un-/zulässig sein so ist das Bit}
    AllowB: Longint;                { des Allow?-Feldes 0 bzw. 1}
    AllowD: Byte;
  end;

  PNumberOfInputsOutputs=^TNumberOfInputsOutputs;
  TNumberOfInputsOutputs=packed record
    Inputs :Byte;                         {Anzahl Eingänge}
    Outputs:Byte;                         {Anzahl Ausgänge}
    NameI : Array[0..49,0..40] of char;
    NameO : Array[0..49,0..40] of char;
  end;

  PInputArray = ^TInputArray;
  TInputArray = packed array[1..50] of extended;
  POutputArray = ^TOutputArray;
  TOutputArray = packed array[1..50] of extended;


function GetVPos(H: HWnd): extended;
{Diese Funktion liefert für das Fenster mit dem Handle H den zugehörigen
 Parameter VPos, d. h. den Zulauf zurück}
var i: integer;
begin
  {Fensterliste nach passendem Fenster durchsuchen}
  for i:=1 to nWindows do if LevelWindows[i].HW = H then
    GetVPos := LevelWindows[i].VPos;
end;

function GetVNeg(H: HWnd): extended;
{Diese Funktion liefert für das Fenster mit dem Handle H den zugehörigen
```

```
 Parameter VNeg, d. h. den Zulauf zurück}
var i: integer;
begin
  {Fensterliste nach passendem Fenster durchsuchen}
  for i:=1 to nWindows do if LevelWindows[i].HW = H then
    GetVNeg := LevelWindows[i].VNeg;
end;

function GetHeight(H: HWnd): extended;
{Diese Funktion liefert für das Fenster mit dem Handle H den zugehörigen
 Parameter Height, d. h. die Füllhöhe zurück}
var i: integer;
begin
  {Fensterliste nach passendem Fenster durchsuchen}
  for i:=1 to nWindows do if LevelWindows[i].HW = H then
    GetHeight := LevelWindows[i].Height;
end;

procedure SetInputs(H: HWnd; VP, VN, Hgt: extended);
{Setzt in der Fensterliste die Eingangswerte des Fensters mit Handle H}
var i: integer;
begin
  for i:=1 to nWindows do if LevelWindows[i].HW = H then begin
    LevelWindows[i].VPos := VP;     {Zulauf}
    LevelWindows[i].VNeg := VN;     {Ablauf}
    LevelWindows[i].Height := Hgt;  {Füllhöhe}
  end;
end;

procedure Paint(DC: HDC; VP, VN, Hgt: extended);
(* Übernimmt die eigentliche Zeichnung des Ausgabefensters *)
var MemDC: HDC;
    RedBrush, BlueBrush, GreenBrush: HBrush;
    HBM: HBitmap;
    BlockBitmap: TBitmap;
    Rect: TRect;
begin
  MemDC := CreateCompatibleDC(DC);
  HBM := LoadBitmap(HInstance, 'BEHAELTER_BITMAP');
  SelectObject(MemDC, HBM);
  GetObject(HBM, SizeOf(BlockBitmap), @BlockBitmap);
  RedBrush := CreateSolidBrush(RGB(255, 0, 0));
  BlueBrush := CreateSolidBrush(RGB(0, 0, 255));
  GreenBrush := CreateSolidBrush(RGB(0, 255, 0));
  {Zulauf}
  with Rect do begin
    Left := 134;
    Right := Left + round(VP*57);
    Top := 17;
    Bottom := 33;
    FillRect(MemDC, Rect, GreenBrush);
  end;
  {Ablauf}
  with Rect do begin
    Left := 168;
    Right := Left + round(VN*57);
    Top := 327;
    Bottom := 343;
    FillRect(MemDC, Rect, RedBrush);
  end;
  {Füllstand}
  with Rect do begin
```



```
    Left := 71;
    Right := 223;
    Bottom := 316;
    Top := Bottom - round(Hgt*200);
    FillRect(MemDC, Rect, BlueBrush);
  end;
  BitBlt(DC, 0, 0, BlockBitmap.bmwidth, BlockBitmap.bmheight, MemDC, 0, 0,
         SrcCopy);
  DeleteObject(RedBrush);
  DeleteObject(GreenBrush);
  DeleteObject(BlueBrush);
  DeleteDC(MemDC);
  DeleteObject(HBM);
end;

function WndFunc(Wnd: HWnd; Msg, wParam: word; lParam: longint): longint;
                                                                    stdcall;
{Fensterfunktion des Ausgabefensters}
var PaintStruct: TPaintStruct;
    DC: HDC;
    i, Index: integer;
begin
  case Msg of
    wm_Paint: begin
                DC := BeginPaint(Wnd, PaintStruct);
                Paint(DC, GetVPos(Wnd), GetVNeg(Wnd), GetHeight(Wnd));
                EndPaint(Wnd, PaintStruct);
              end;
    wm_Destroy: begin
                  {Das Fenster soll zerstört werden. Wir müssen es also
                   aus der Fensterliste löschen und alle anderen Einträge
                   um eins nach vorn verschieben}
                  Index := 0;
                  repeat inc(Index) until LevelWindows[Index].HW = Wnd;
                  for i:=Index+1 to nWindows do
                    LevelWindows[i-1] := LevelWindows[i];
                  dec(nWindows);
                end;
  end;
  WndFunc := DefWindowProc(Wnd, Msg, wParam, lParam);
end;

const
  WindowClass: TWndClass = (
    style: 0;
    lpfnWndProc: @WndFunc;
    cbClsExtra: 0;
    cbWndExtra: 0;
    hInstance: 0;
    hIcon: 0;
    hCursor: 0;
    hbrBackground: 1;
    lpszMenuName: nil;
    lpszClassName: 'LevelWindow');

procedure GetParameterStruct(D:PParameterStruct); export stdcall;
begin
  (* keine Parameter erforderlich ! *)
  D^.NuE := 0;
  D^.NuI := 0;
  D^.NuB := 0;
end;
```



```
procedure GetDialogEnableStruct(D:PDialogEnableStruct;D2:PParameterStruct);
                                                              export stdcall;
begin
  (* wird hier nicht benötigt, da kein Dialog erforderlich *)
end;

procedure GetNumberOfInputsOutputs(D:PNumberOfInputsOutputs);export stdcall;
begin
  D^.Inputs:=3;    {Unser Block hat drei Eingänge...}
  D^.Outputs:=0;   {            und keinen Ausgang  }
  {Namen der iIngänge}
  StrCopy(D^.NameI[0], 'Zulauf');
  StrCopy(D^.NameI[1], 'Ablauf');
  StrCopy(D^.NameI[2], 'Füllstand');
end;

procedure InitUserDLL(D:PParameterStruct); export stdcall;
{Diese Prozedur wird vom User-DLL-Block aufgerufen, wenn er initialisiert
 wird. Wir benötigen ein Flag, an dem wir später erkennen können,
 ob das Ausgabefenster schon existiert. Dazu nehmen wir den ersten
 Integer-Parameter D^.I[0]. Im Prinzip könnten wir das Ausgabefenster auch
 schon hier erzeugen; wir wollen aber, dass es erst bei Start der ersten
 Simulation erscheint; dadurch wird die ganze Sache etwas komplizierter!}
begin
  D^.I[0] := 0;
end;

procedure DisposeUserDLL(D:PParameterStruct); export stdcall;
{Diese Prozedur wird vom User-DLL-Block aufgerufen, wenn der Block gelöscht
 wird oder eine andere DLL für den Block geladen wird. In diesen Fällen
 müssen wir das Ausgabefenster löschen}
begin
  DestroyWindow(D^.UserHWindow);
end;

function CanSimulateDLL(D:PParameterStruct):Integer; export stdcall;
begin
  CanSimulateDLL:=1;  {Simulation immer möglich}
end;

procedure InitSimulationDLL(D:PParameterStruct;Inputs:PInputArray;
                            Outputs:POutputArray); export stdcall;
var Rect: TRect;
    x, y: integer;
begin
  with Rect do begin
    Top := 0;
    Bottom := cyBitmap;
    Left := 0;
    Right := cxBitmap;
  end;
  (* Fenstergröße an Bitmapgröße anpassen! *)
  AdjustWindowRect(Rect, ws_Popup or ws_Caption, false);
  {Falls noch kein Ausgabefenster existiert, wird es jetzt angelegt...}
  if D^.I[0] = 0 then begin
    D^.I[0] := 1; {kennzeichnen, dass Fenster jetzt vorhanden}
    {Damit bei mehreren Fenster nicht alle aufeinander liegen, verschieben
     wir jedes um 50 Pixel nach links oben}
    x := 350 + nWindows*50;
    y := 70 - nWindows*50;
    {Es ist soweit: Das Fenster wird erzeugt...}
```



```
    D^.UserHWindow := CreateWindowEx(ws_ex_TopMost, 'LevelWindow', '',
                              ws_Popup or ws_Caption or ws_MinimizeBox,
                              x, y, Rect.Right-Rect.Left, Rect.Bottom-
                              Rect.Top, D^.ParentHWnd, 0, HInstance, nil);
    {...angezeigt...}
    ShowWindow(D^.UserHWindow, sw_Show);
    {...und in die Fensterliste eingetragen!}
    inc(nWindows);
    LevelWindows[nWindows].HW := D^.UserHWindow;
  end;
  {Wir wollen dem Fenster den Namen des User-DLL-Blocks geben}
  SetWindowText(D^.UserHWindow, D^.ParentName);
end;

procedure SimulateDLL(T:Extended;D:PParameterStruct;Inputs:PInputArray;
                      Outputs:POutputArray);export stdcall;
var DC: HDC;
begin
  {Inputs in Fensterliste eintragen...}
  SetInputs(D^.UserHWindow, Inputs^[1], Inputs^[2], Inputs^[3]);
  {...und Fensterinhalt neu zeichnen}
  DC := GetDC(D^.UserHWindow);
  Paint(DC, GetVPos(D^.UserHWindow), GetVNeg(D^.UserHWindow),
        GetHeight(D^.UserHWindow));
  ReleaseDC(D^.UserHWindow, DC);
end;

procedure EndSimulationDLL; export stdcall;
begin
  (* hier nicht erforderlich *)
end;

function SetInputChar: PChar; export stdcall;
begin
  {Ersten Eingang mit '+', zweiten mit '-' und dritten mit 'H' beschriften}
  SetInputChar := '+-H';
end;

procedure ShowWindowDLL(D: PParameterStruct); export stdcall;
begin
  {Anzeigefenster in Normalgröße anzeigen}
  ShowWindow(D^.UserHWindow, sw_Normal);
end;

procedure HideWindowDLL(D: PParameterStruct); export stdcall;
begin
  {Anzeigefenster zum Symbol verkleinern}
  ShowWindow(D^.UserHWindow, sw_Minimize);
end;

procedure IsUserDLL32; export stdcall;
begin
end;

exports
  InitUserDLL, DisposeUserDLL, GetParameterStruct,
  GetDialogEnableStruct, GetNumberOfInputsOutputs,
  CanSimulateDLL, InitSimulationDLL, SimulateDLL,
  EndSimulationDLL, SetInputChar, ShowWindowDLL,
  HideWindowDLL, IsUserDLL32;

begin
```



```
  (* Fensterklasse registrieren *)
  WindowClass.hInstance := HInstance;
  WindowClass.hIcon := LoadIcon(0, idi_Application);
  WindowClass.hCursor := LoadCursor(0, idc_Arrow);
  RegisterClass(WindowClass);
  nWindows := 0;
end.
```

Pascal-Listing zu Beispiel 3

Weitere Beispiele

Im Lieferumfang von BORIS befindet sich eine Vielzahl weiterer User-DLLs, die in der Regel in Pascal (DELPHI 3) entwickelt wurden. Die fertigen DLLs finden Sie im Unterverzeichnis *UserDLLs*, die zugehörigen Quelltexte - sofern vorhanden - im Unterverzeichnis *UserDLLs\Sources*. Einige dieser User-DLLs lassen sich direkt über die Palettenseite *User* der Systemblock-Toolbar erreichen.

Palettenseite User *der Systemblock-Toolbar*

Hinweise zur Programmierung unter Visual C++

Bei der Programmierung von User-DLLs unter *Microsoft Visual C++* ist - zumindest in der bei der Drucklegung dieser Dokumentation aktuellen Version 6 - eine Problematik zu beachten, die mit dem im Rahmen der User-DLL-Schnittstelle benutzten 10-Byte-Fließkomma-Datentyp (*extended* in Pascal bzw. *long double* in C) zusammenhängt. Die Firma Microsoft hat dem Datentyp *long double* nämlich intern nicht eine Länge von zehn Byte, sondern entgegen der üblichen Konvention nur eine Länge von acht Byte zugeordnet. Verwendet man daher diesen Datentyp direkt bei der Programmierung von User-DLLs, ergeben sich später bei der Arbeit mit diesen DLLs innerhalb von BORIS mit an Sicherheit grenzender Wahrscheinlichkeit Probleme.

Dennoch lassen sich mit Visual C++ natürlich ohne weiteres User-DLLs für BORIS erstellen. Dazu ist lediglich ein kleiner Workaround notwendig, dessen Kern darin besteht, einen eigenen "Hilfs"-Datentyp zu definieren, der tatsächlich 10 Byte lang ist, und alle *long double*-Deklarationen innerhalb der User-DLL-Schnittstelle durch diesen Datentyp zu ersetzen. Es sind dann zusätzlich

nur noch zwei Umrechnungsroutinen zu implementieren, die den 10-Byte-Datentyp in einen 8-Byte-Fließkommawert vom Typ *double* umwandeln bzw. umgekehrt. Diese werden dann innerhalb der User-DLL-Funktionen jeweils zur Umrechnung der Datentypen aufgerufen.

Im Unterverzeichnis *UserDLLs\Sources\C++\VisualC++* finden Sie ein Visual C++-Projekt namens CPPDEMO.CPP, welches das weiter oben beschriebene *Beispiel 1: Eine simple User-DLL* nach diesem Schema implementiert. Der Quellcode ist so erstellt worden, dass er sowohl mit Visual C++ als auch z. B. mit dem Borland C++-Builder compilierbar ist. Dazu wurde der Datentyp *NUMType* eingeführt, der den oben beschriebenen 10-Byte-Datentyp realisiert. Nachfolgende Listings zeigen den Quellcode der entsprechenden Header- bzw. C-Dateien.

```
#ifndef WFNumType
#define WFNumType WFNumType

// Diese Headerdatei bietet ein Workaround für VC++ an.
// In VC++ ist der Typ 'long double' auf 'double' umgemünzt,
// was nach ANSI-C nicht verkehrt ist,
// aber schade, denn der Fließkommaprozessor arbeitet intern
// mit der maximalen Genauigkeit,
// z.Zt. 80 Bit (also 10 Byte) und nicht 8 Byte.

// BORIS verwendet aber 10 Byte große Fließpunktzahlen, was bei
// VC++ zu einem Problem führt.

// Lösung:
//   1.) Verwenden Sie  ÜBERALL  wo normalerweise 'long
//       double' stehen
//       müsste, den hier deklarierten Typ 'NUMType' !
//   2.) Zur Umrechnung von 'NUMType' zu 'double' und
//       umgekehrt stehen Ihnen entsprechend benannte (sehr
//       schnelle) Funktionen zur Verfügung.

// Beispiel:
//
//  long double f(long double a)
//  {
//      a+=1;
//      return a;
//  }
//
// wird zu:
//
//  NUMType f(NUMType a)
//  {
//      double d=NUMType2double(a);
//      d+=1;
//      return double2NUMType(d);
//  }
//
```



```
// Diese Headerdatei setzt alles auf den normalen Zustand
// (Also NUMType auf long double und die Umrechnungsfunktionen
//  auf das direkte Argument), falls es sich nicht um VC++
// handelt. Somit kann sie
// für Compiler mit und ohne diesen Unterschied verwendet
// werden.


#ifndef  _MSC_VER

#define NUMType long double
#define NUMType2double(a) a
#define double2NUMType(a) a

#else
#define MS_LONG_DOUBLE_PROBLEM


typedef struct{
    char d[10];
} NUMType;



inline double NUMType2double(const NUMType mem)
{
    double d;
    _asm fld TBYTE PTR mem
    _asm fstp QWORD PTR d
    return d;
}

inline NUMType double2NUMType(const double d)
{
    NUMType mem;
    _asm fld QWORD PTR d
    _asm fstp TBYTE PTR mem
    return mem;
}


#endif

#endif
```

Listing zu WFNumType.h

```
#ifndef cppdemo
#define cppdemo cppdemo

#pragma pack(1)

/***********************************************
 *                                             *
 *       Einfache Demo-DLL für BORIS           *
```

```
 *       (Beispiel 1 des Handbuchs)              *
 *                                               *
 ************************************************/

#ifdef _MSC_VER
#define WFCallingConvention  __stdcall
#elif  __BORLANDC__
#define WFCallingConvention  _stdcall _export
#endif


typedef struct{
    char    NuE;
    char    NuI;
    char    NuB;
    NUMType E[32];        // ersetzt long double-Datentyp!!!
    int     I[32];
    char    B[32];
    char    D[256];
    NUMType EMin[32];     // ersetzt long double-Datentyp!!!
    NUMType EMax[32];     // ersetzt long double-Datentyp!!!
    int     IMin[32];
    int IMax[32];
    char NaE[32][41];
    char NaI[32][41];
    char NaB[32][41];
   /*den Rest der Variablen von TParameterStruct können wir uns
     sparen, da wir ihn nicht benötigen!  */
  } TParameterStruct, *PParameterStruct;

typedef struct {
    long AllowE;
    long AllowI;
    long AllowB;
    char AllowD;
  } TDialogEnableStruct, *PDialogEnableStruct;


typedef struct {
    char Inputs;                //Anzahl Eingänge
    char Outputs;               //Anzahl Ausgänge
    char NameI[50][41];
    char NameO[50][41];
  } TNumberOfInputsOutputs, *PNumberOfInputsOutputs;

typedef NUMType TInputArray[10]; //ersetzt long double-Datentyp
typedef NUMType TOutputArray[10];//ersetzt long double-Datentyp

#pragma pack()

extern "C"
{
 void WFCallingConvention
   GetParameterStruct(PParameterStruct);
```



```
 void WFCallingConvention
   GetDialogEnableStruct(PDialogEnableStruct,PParameterStruct);
 void  WFCallingConvention
   GetNumberOfInputsOutputs(PNumberOfInputsOutputs D);
 int   WFCallingConvention
   CanSimulateDLL(PParameterStruct D);
 void  WFCallingConvention
   InitSimulationDLL(PParameterStruct D, TInputArray Inputs,
                     TOutputArray Outputs);
 void  WFCallingConvention
   SimulateDLL(NUMType T,PParameterStruct D,
               TInputArray Inputs, TOutputArray Outputs);
 char* WFCallingConvention SetInputChar(void);
 void  WFCallingConvention EndSimulationDLL(void);
 void  WFCallingConvention IsUserDLL32(void);
}

#endif
```

Listing zu CPPDemo.h

```
#include <string.h>
#include "WFNUMType.h"
#include "cppdemo.h"

void WFCallingConvention GetParameterStruct(PParameterStruct D)
{
  D->NuE=3;
  D->NuI=0;
  D->NuB=1;
  strncpy(D->NaE[0],"Verstärkung k1 für Steuereing. < 0",39);
  strncpy(D->NaE[1],"Verstärkung k2 für Steuereing. = 0",39);
  strncpy(D->NaE[2],"Verstärkung k3 für Steuereing. > 0",39);
  strncpy(D->NaB[0],"Ausgang invertiert",39);
  D->E[0]=double2NUMType(1);
  D->E[1]=double2NUMType(5);
  D->E[2]=double2NUMType(10);
  D->B[0]=0;
  D->EMin[0]=double2NUMType(0.0);
  D->EMax[0]=double2NUMType(100000);
  D->EMin[1]=double2NUMType(0.0);
  D->EMax[1]=double2NUMType(100000);
  D->EMin[2]=double2NUMType(0.0);
  D->EMax[2]=double2NUMType(100000);
}


void WFCallingConvention
  GetDialogEnableStruct(PDialogEnableStruct D,
                        PParameterStruct D2)
{
//Alle Dialogelemente bis auf Dateien sollen jederzeit
zugänglich sein
  D->AllowE=0xFFFFFFFF;
  D->AllowI=0xFFFFFFFF;
```



```
  D->AllowB=0xFFFFFFFF;
}


void  WFCallingConvention
  GetNumberOfInputsOutputs(PNumberOfInputsOutputs D)
{ D->Inputs=2;
  D->Outputs=1;
  strncpy(D->NameI[0],"Steuereingang",39);
  strncpy(D->NameI[1],"Dateneingang",39);
  strncpy(D->NameO[0],"Ausgang",39);
}


int  WFCallingConvention CanSimulateDLL(PParameterStruct D)
{ //Simulation immer zulässig!
  return 1;
}

void WFCallingConvention
  SimulateDLL(NUMType T,PParameterStruct D,
              TInputArray Inputs,TOutputArray Outputs)
{
  double I0=NUMType2double(Inputs[0]);
  if (I0<0) //Eingang 1 negativ
    Outputs[0] = double2NUMType(NUMType2double(D->E[0]) *
                                NUMType2double(Inputs[1]));
  else if (I0 == 0) //Eingang 1 null
    Outputs[0] = double2NUMType(NUMType2double(D->E[1]) *
                                NUMType2double(Inputs[1]));
  else //Eingang 1 positiv
    Outputs[0] = double2NUMType(NUMType2double(D->E[2]) *
                                NUMType2double(Inputs[1]));
  if (D->B[0] == 1) //Ausgang invertieren
    Outputs[0] = double2NUMType(-NUMType2double(Outputs[0]));
}


void WFCallingConvention
  InitSimulationDLL(PParameterStruct D,
                    TInputArray Inputs,TOutputArray Outputs)
{/*Der Initialisierungsschritt soll genauso durchgeführt werden
   wie alle Simulationsschritte. Also rufen wir einfach
   SimulateDLL auf!*/
 SimulateDLL(double2NUMType(0), D, Inputs, Outputs);
}

void WFCallingConvention EndSimulationDLL(void)
{//wird nicht benötigt
}

char* WFCallingConvention SetInputChar(void)
{
  return "SD";
```

```
}

void WFCallingConvention IsUserDLL32(void)
{
}
```

Listing zu CPPDemo.cpp

Benutzerdefinierte Block-Bitmaps

Optional hat der Anwender die Möglichkeit, jeden seiner User-Blöcke mit einem *anwenderspezifischen Block-Bitmap* zu versehen (siehe auch Abschnitt *Arbeiten mit Superblöcken*). Dieses Block-Bitmap wird als BMP-Datei erstellt und muss den selben Namen aufweisen wie die DLL, jedoch mit der Extension BMP, und sich im selben Verzeichnis befinden wie die DLL. Zu einer DLL mit dem Namen MOTOR.DLL gehört also die Bitmap-Datei MOTOR.BMP. BORIS überprüft beim Einfügen eines User-Blocks automatisch, ob die zugehörige BMP-Datei vorhanden ist. Ist dies der Fall, wird diese benutzt, ansonsten das Standard-Userblock-Bitmap. Das Bitmap sollte eine Größe von 48 ×42 Pixeln aufweisen; Bitmaps anderer Größe werden automatisch gedehnt bzw. gestaucht, so dass sie optimal in das Blockschaltbild passen. Auch für die Druckerausgabe kann ein benutzerdefiniertes Bitmap (möglichst als s/w-Bitmap) definiert werden. Dieses muss am Ende des Dateinamens zusätzlich die Kennung *_P* erhalten. Soll die User-DLL auch auf der Palettenseite *User* der Systemblock-Toolbar erscheinen, muss zusätzlich ein 18x18-Pixel großes Bitmap definiert werden, welches die Kennung *_T* enthält.

Beispiel:	zu MOTOR.DLL	gehört Drucker-Bitmap	MOTOR_P.BMP
		und Toolbar-Bitmap	MOTOR_T.BMP

Entwurf von PID-Reglern

Neben der reinen Simulation von (Regelungs-) Systemen bietet BORIS Ihnen die Möglichkeit zum direkten Entwurf von PID-Reglern. Dabei haben Sie zwei Möglichkeiten:

- Entwurf von PID-Reglern auf der Basis von sogenannten *Einstellregeln*. Diese Entwurfsmethodik ist Bestandteil dieses Kapitels.
- Entwurf von PID-Reglern durch *numerische Parameteroptimierung* mit Hilfe des *BORIS-Parameter-Optimierungsmoduls* (siehe Abschnitt *Numerische Optimierung von Systemen*)

PID-Entwurf nach Einstellregeln

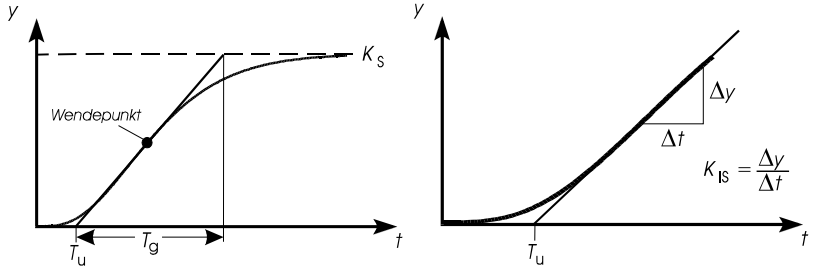
Kenngrößen der Regelstrecke

Die Grundlage des in BORIS realisierten PID-Entwurfs nach Einstellregeln bildet die *Sprungantwort* der Regelstrecke, d. h. der Verlauf der Ausgangsgröße der Strecke bei einem Sprung der Höhe eins an ihrem Eingang. Aus dieser Sprungantwort lassen sich dann *Verzugszeit* T_u und *Ausgleichszeit* T_g der Strecke (bei Strecken mit Ausgleich) bzw. nur die Verzugszeit (bei Strecken ohne Ausgleich) ermitteln. Zusammen mit der *Streckenverstärkung* (*Übertragungsbeiwert*) K_S bzw. K_{IS} bilden diese Parameter dann die Grundlage des Reglerentwurfs (siehe nachfolgendes Bild; näheres dazu z. B. in [7]). In der Regel lassen sich die Einstellregeln nur auf nicht schwingfähige Strecken anwenden.

Hinweis: Einstellregeln für "Nicht-Standard-Strecken" befinden sich zur Zeit in Vorbereitung und werden in einer späteren Version verfügbar sein. Weitere Einstellregeln (z. B. die *T-Summen-Regel*) sind als optionale Module zu BORIS erhältlich.

Die Ermittlung der Kenngrößen in BORIS kann manuell oder automatisch erfolgen. Dazu ist zunächst die durch Messung aufgenommene, durch Simula-

tion erzeugte oder aus einer externen Datei gelesene Sprungantwort einem Zeitverlauf-Block zuzuführen.



Kenngrößen der Strecke für Strecken mit Ausgleich (links) bzw. ohne Ausgleich (rechts)

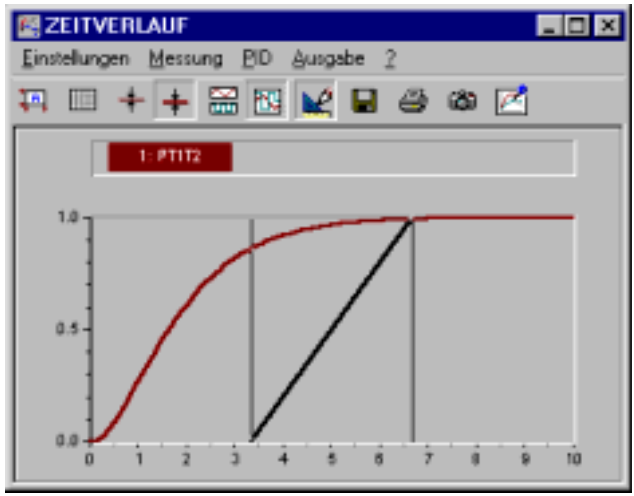
Soll die Kennwertermittlung manuell erfolgen, so muss die Messfunktion des Zeitverlauf-Ausgabefensters aktiviert werden. Außerdem muss über seine Menüoption MESSUNG | TANGENTE EINZEICHNEN die Einzeichnung der Tangente aktiviert werden. Dadurch erscheint im Anzeigefenster zusätzlich zu den Messcursorn eine Tangente, die ebenfalls mit der Maus verschoben werden kann:

- Zunächst ist die Tangente an die Messcursor "geheftet" und lässt sich mit diesen verschieben.
- Durch gleichzeitige Betätigung der <Shift>- oder <Strg>-Taste lassen sich die Messcursor bei Bedarf auch von der Tangente "lösen".

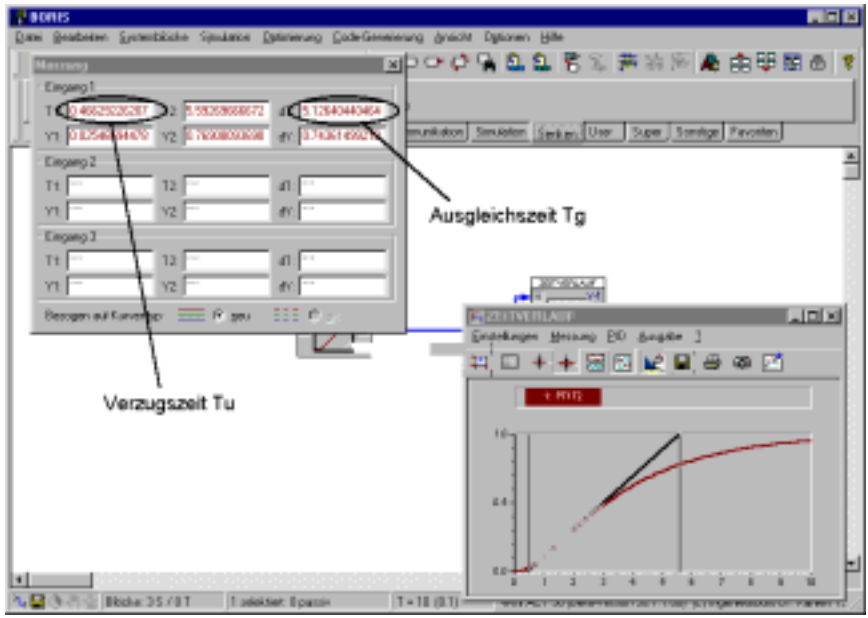
Auf diese Weise lässt sich die Tangente manuell als Wendetangente an die Sprungantwort anlegen. Aus dem Messwertdialog lassen sich dann die Werte für Verzugszeit bzw. Ausgleichszeit entnehmen (siehe nachfolgendes Bild).

Zur automatischen Kennwertermittlung wird der Menüpunkt PID| PID-ENTWURF... angewählt. Man gelangt dann in den kombinierten Analyse-/Entwurfsdialog. Das obere Gruppenfeld mit dem Titel *Streckenparameter* ist für die Kennwertermittlung maßgebend.

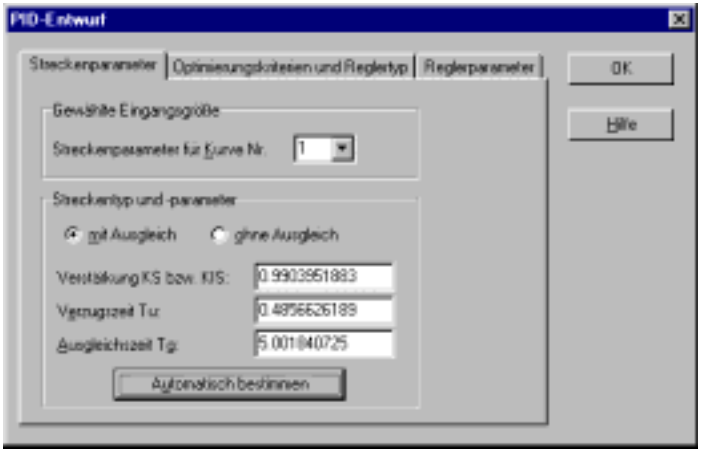
Über das Editierfeld *Streckenparameter für Kurve Nr.* kann bei mehreren dargestellten Kurven zunächst die zu berücksichtigende Sprungantwort ausgewählt werden. Die Kennwertermittlung wird dann über den Schalter *Automatisch bestimmen* gestartet. BORIS ermittelt daraufhin zunächst den Streckentyp (mit/ohne Ausgleich) und im Anschluss daran die Kennwerte der Strecke. Alle Größen können bei Bedarf auch manuell modifiziert werden.



Anzeigefenster des Zeitverlauf-Blocks nach Aktivierung der Messfunktion und der Tangente



Manuelle Ermittlung von Verzugs- und Ausgleichszeit



Analyse- bzw. Entwurfsdialog (hier nach erfolgter Kennwertermittlung)

Ermittlung der Reglerparameter

Nach erfolgter Kennwertermittlung kann der Reglerentwurf gestartet werden. Dazu wird - falls nicht bereits geschehen - über PID | PID-ENTWURF... in den Analyse-/Entwurfsdialog gewechselt. Zum Entwurf werden die Einstellregeln nach Chien/Hrones und Reswick benutzt (siehe nachfolgende Tabellen). Beim Entwurf ist anzugeben, ob ein Überschwingen der Ausgangsgröße zugelassen werden soll oder nicht und ob besonders gutes Führungs- oder aber Störverhalten erzielt werden soll. Diese Optimierungskriterien und der zu entwerfende Reglertyp sind auf der Palette *Optimierungskriterien und Reglertyp* festzulegen.

Der eigentliche Entwurfsvorgang wird über den Schalter *Reglerparameter berechnen* auf der Palette *Reglerparameter* gestartet. Die ermittelten Reglerparameter werden dann in den entsprechenden Editierfeldern im Gruppenfeld *Reglerparameter* angezeigt und können dort ggf. modifiziert werden. Ist bereits ein PID-Block vorhanden, so können die ermittelten Parameter über den Schalter *Parameter übernehmen* direkt in den ausgewählten PID-Block übernommen werden.

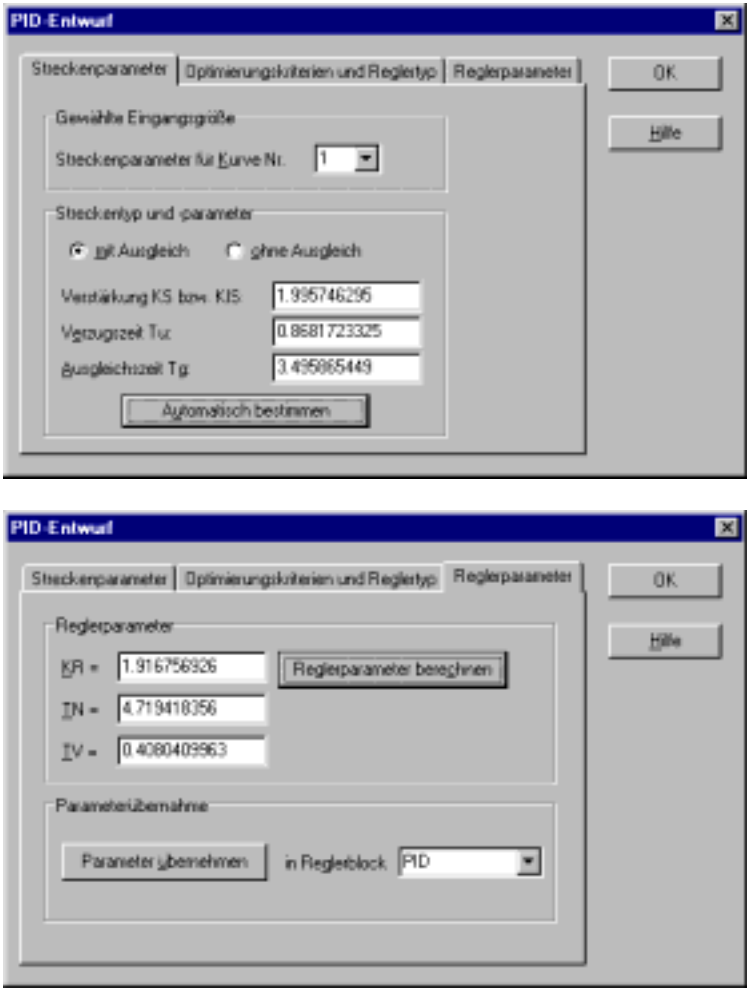
Regler-typ	Mit Überschwingen		Ohne Überschwingen	
	Störung	**Führung**	**Störung**	**Führung**
P	$K_R = 0.71\frac{1}{K_S}\frac{T_g}{T_u}$	$K_R = 0.71\frac{1}{K_S}\frac{T_g}{T_u}$	$K_R = 0.3\frac{1}{K_S}\frac{T_g}{T_u}$	$K_R = 0.3\frac{1}{K_S}\frac{T_g}{T_u}$
PI	$K_R = 0.71\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 2.3T_u$	$K_R = 0.59\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = T_g$	$K_R = 0.59\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 4T_u$	$K_R = 0.34\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 1.2T_g$
PID	$K_R = 1.2\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 2T_u$ $T_V = 0.42T_u$	$K_R = 0.95\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 1.35T_g$ $T_V = 0.47T_u$	$K_R = 0.95\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 2.4T_u$ $T_V = 0.42T_u$	$K_R = 0.59\frac{1}{K_S}\frac{T_g}{T_u}$ $T_N = 1.35T_g$ $T_V = 0.5T_u$

Regler-typ	Mit Überschwingen		Ohne Überschwingen	
	Störung	**Führung**	**Störung**	**Führung**
P	$K_R = 0.71\frac{1}{K_{IS}}\frac{1}{T_u}$	$K_R = 0.71\frac{1}{K_{IS}}\frac{1}{T_u}$	$K_R = 0.3\frac{1}{K_{IS}}\frac{1}{T_u}$	$K_R = 0.3\frac{1}{K_{IS}}\frac{1}{T_u}$
PI	$K_R = 0.71\frac{1}{K_{IS}}\frac{1}{T_u}$ $T_N = 2.3T_u$	$K_R = 0.59\frac{1}{K_{IS}}\frac{1}{T_u}$	$K_R = 0.59\frac{1}{K_{IS}}\frac{1}{T_u}$ $T_N = 4T_u$	$K_R = 0.34\frac{1}{K_{IS}}\frac{1}{T_u}$
PID	$K_R = 1.2\frac{1}{K_{IS}}\frac{1}{T_u}$ $T_N = 2T_u$ $T_V = 0.42T_u$	$K_R = 0.95\frac{1}{K_{IS}}\frac{1}{T_u}$ $T_V = 0.47T_u$	$K_R = 0.95\frac{1}{K_{IS}}\frac{1}{T_u}$ $T_N = 2.4T_u$ $T_V = 0.42T_u$	$K_R = 0.59\frac{1}{K_{IS}}\frac{1}{T_u}$ $T_V = 0.5T_u$

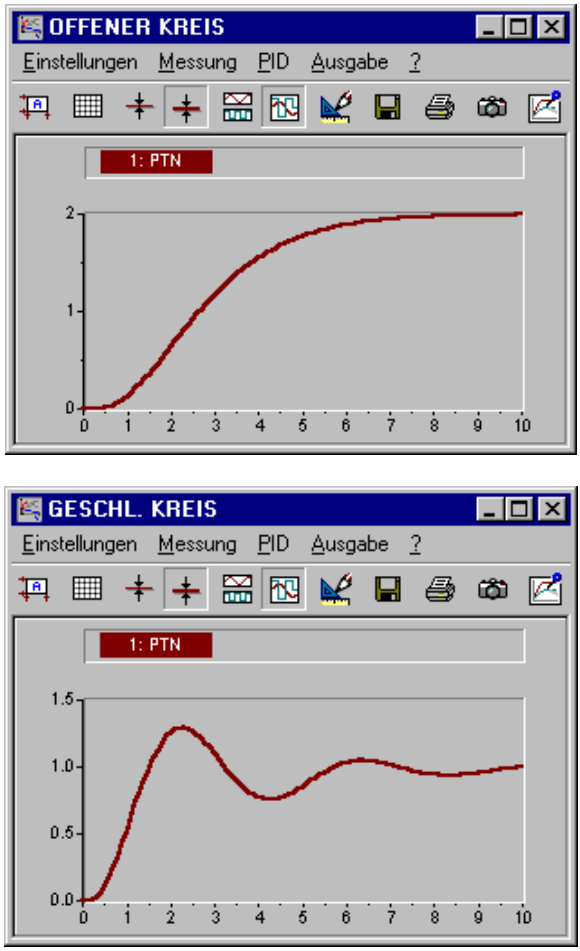
Einstellregeln nach Chien/Hrones und Reswick für Strecken mit (oben) bzw. ohne Ausgleich (unten)

Beispiel

Es soll für eine PT_3-Strecke mit einer dreifachen Zeitkonstanten von 1 und einer Verstärkung von 2 ein PID-Regler entworfen werden, der für gutes Führungsverhalten sorgt. Ein Überschwingen der Regelgröße soll akzeptiert werden. Nachfolgende Grafiken zeigen den Entwurfsdialog nach der Streckenanalyse und dem Reglerentwurf sowie die Sprungantworten der Strecken und des resultierenden geschlossenen Regelkreises.



Streckenanalyse (oben) und Reglerentwurf (unten) für PT_3-Beispielstrecke

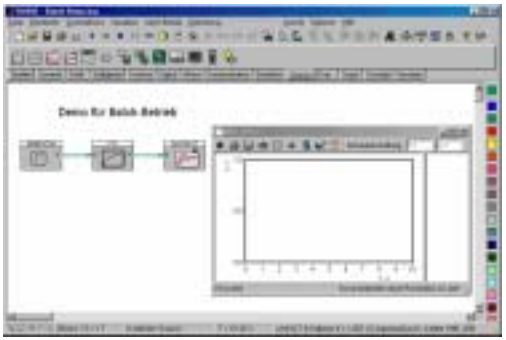


Sprungantworten der Regelstrecke (oben) bzw. des geschlossenen Regelkreises (unten)

Hinweis: Das Beispiel befindet sich unter dem Namen PID_DESI.BSY im WinFACT-Beispielverzeichnis.

Simulationen im Batch-Betrieb

Neben einzelnen Simulationsläufen ermöglicht BORIS auch die automatische Durchführung von Simulationsreihen, beispielsweise zur Untersuchung der Auswirkung einzelner Parameter auf das Systemverhalten. Dieser Batch-Betrieb basiert auf dem Prinzip der Exportparameter, d. h. alle zum Export freigeschalteten Parameter der aktuellen Systemstruktur können innerhalb eines Batch-Laufes modifiziert werden. Das Prinzip des Batch-Betriebs soll hier an einem einfachen Beispiel erläutert werden. Dazu soll die Sprungantwort eines PT_1-Gliedes in Abhängigkeit von der Verstärkung *K* und der Zeitkonstanten *T* des Gliedes untersucht werden.

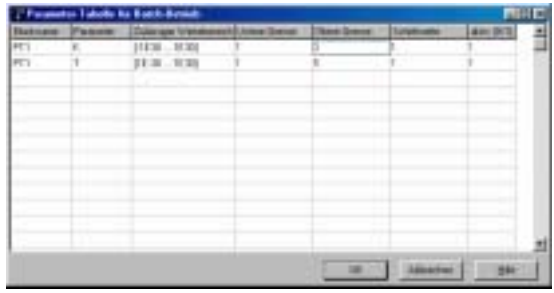


Demo-Struktur für Batch-Betrieb (Datei BATCH-DEMO.BSY*) vor dem Start des Batch-Laufes*

Gehen Sie zur Einrichtung des Batch-Betriebs wie folgt vor (die fertige Struktur finden Sie unter dem Namen BATCH-DEMO.BSY im WinFACT-Beispielverzeichnis):

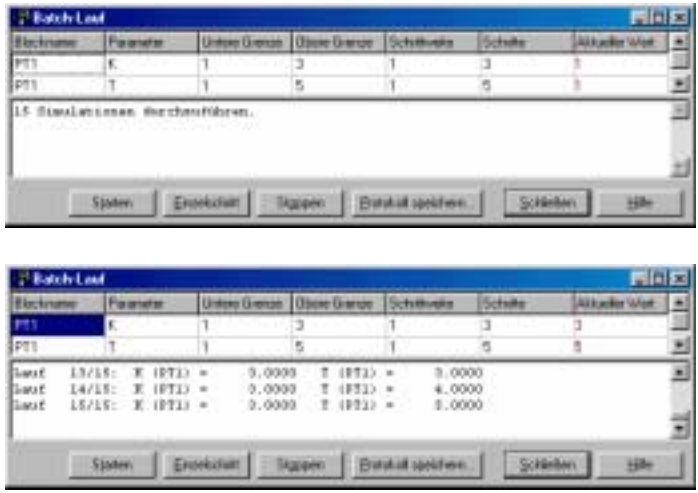
- Erstellen Sie die Systemstruktur wie oben gezeigt bestehend aus Generator, PT_1-Glied und Mehrfach-Plotter.
- Geben Sie die Parameter des PT_1-Gliedes zum Export frei.
- Wählen Sie nun die Menüoption BATCH-BETRIEB | PARAMETER-TABELLE.... Sie gelangen dadurch in den Dialog zur Konfigurierung der Parameter. Wir wollen für die Verstärkung *K* des PT_1-Glieds Werte von 1, 2 und 3 wählen und für die Zeitkonstante *T* Werte von 1, 2, 3, 4 und 5s. Geben Sie die Parameter daher wie unten gezeigt in den Dialog ein. Da-

mit auch tatsächlich eine Variation der Parameter stattfindet, müssen beide Parameter noch durch eine 1 in der letzten Spalte aktiviert werden.



Konfigurierung der Parameter für den Batch-Betrieb

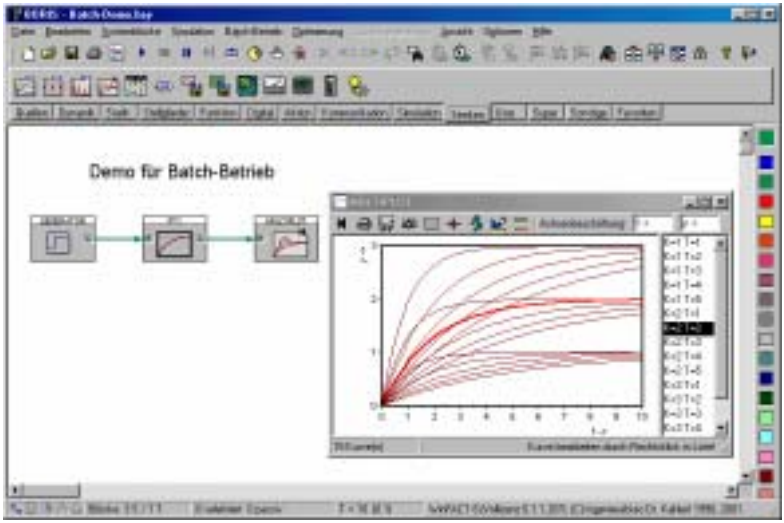
➢ Verlassen Sie den Dialog und wählen Sie die Menüoption BATCH-BETRIEB | BATCH-LAUF STARTEN.... Es erscheint nun der eigentliche Dialog für den Batch-Betrieb. Starten Sie den Batch-Lauf über die Schaltfläche *Starten*. Sie können nun innerhalb des Dialogs den Ablauf der eigentlichen Simulationen verfolgen. Bei Bedarf lässt sich ein laufender Batch-Betrieb über die Schaltfläche *Stoppen* stoppen; über die Schaltfläche *Einzelschritt* lässt sich der Batch-Betrieb schrittweise durchführen. Über die Schaltfläche *Protokoll speichern...* kann das im unteren Dialogteil ausgegebene Protokoll in eine Datei gespeichert werden.



Batch-Dialog vor Start des Batch-Laufes (oben) und danach (unten)

- Verlassen Sie nunmehr den Batch-Dialog und betrachten Sie das Ergebnis des Batch-Laufs im Mehrfach-Plotter. Dieser enthält für jede Simulation (d. h. für jede Parameterkombination) eine Sprungantwort; alle Kurven sind in der Liste am rechten Fensterrand automatisch betitelt worden. Bei Anklicken eines Kurventitels wird die entsprechende Kurve farblich hervorgehoben, um eine eindeutige Zuordnung zu kennzeichnen.

Der Mehrfach-Plotter eignet sich also besonders für derartige Simulationsstudien im Batch-Betrieb. Sollen die Ergebnisse der einzelnen Simulationsläufe eines Batch-Laufes zusätzlich oder alternativ in Dateien abgelegt werden, so lassen sich dafür die Blocktypen FILEOUTPUT bzw. TABFILEOUTPUT benutzen; diese können nämlich so parametriert werden, dass bei jedem Simulationslauf automatisch eine neue Datei mit entsprechendem Namen angelegt wird, die zu Beginn als Kommentar die aktuell benutzten Parameterwerte enthält. Einzelheiten dazu entnehmen Sie bitte der Beschreibung der entsprechenden Blocktypen im Abschnitt *Die BORIS-Systemblock-Bibliothek*.

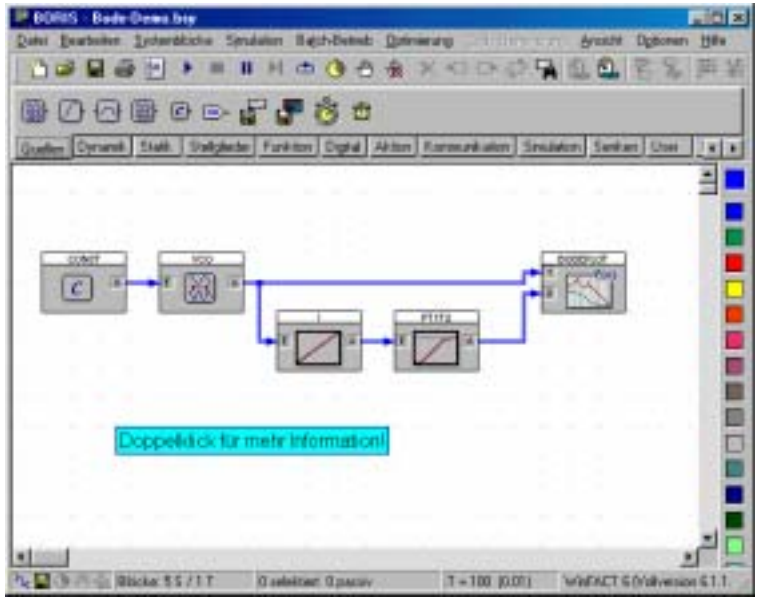


Ergenbis des Batch-Laufs im Mehrfach-Plotter (hier mit angewählter Sprungantwort für K = T = 2)

Frequenzgang-Ermittlung

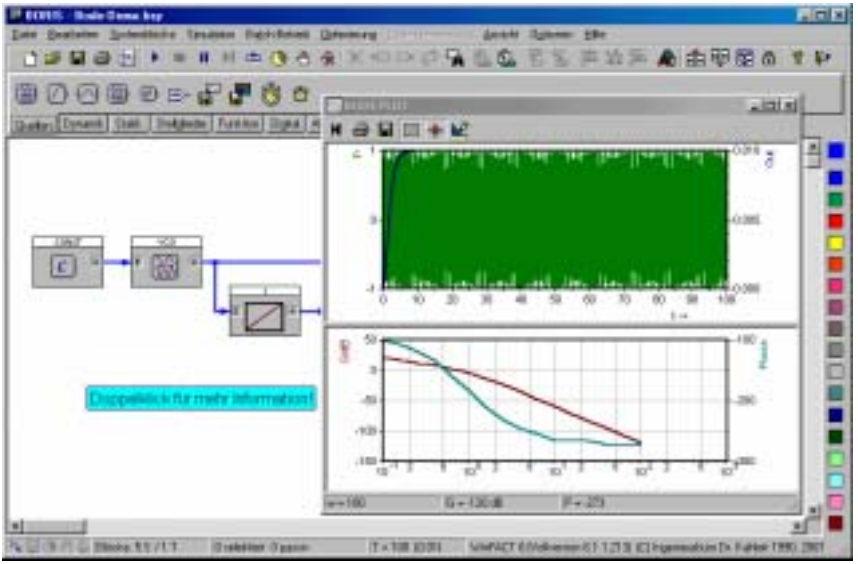
Der Batch-Betrieb kann in Kombination mit dem Frequenzgang-Plotter (Blocktyp BODEPLOT) auch benutzt werden, um den Frequenzgang eines Systems (wahlweise als Bode-Diagramm oder Nyquist-Ortskurve) zu ermitteln. Dazu erstellt man sich zunächst mit Hilfe eines Konstanten-Blocks (Blocktyp CONST) und eines VCOs (Blocktyp VCO) einen *Wobbelgenerator* und regt das zu untersuchende System dann im Batch-Betrieb mit steigenden Frequen-

zen an. Nachfolgende Bildschirmgrafik zeigt dies am Beispiel der Datei BODE-DEMO.BSY aus dem Beispiel-Verzeichnis.



Simulationsstruktur zur Ermittlung des Frequenzgangs (Bode-Diagramm) eines Systems (hier einer IT_2-Strecke)

Die Eingangsgröße des zu untersuchenden Systems wird Eingang 1 des Frequenzgang-Plotters, die Ausgangsgröße Eingang 2 zugeführt. Der VCO sollte zweckmäßigerweise in der Betriebsart *exponentiell* betrieben werden. Die Konstante des CONST-Blocks (d. h. seine Ausgangsgröße) legt dann die aktuelle Frequenz fest und wird per Batch modifiziert. Soll der Frequenzgang z. B. für Frequenzen zwischen $\omega = 0.1$ ($= 10^{-1}$) und $\omega = 100$ ($= 10^{2}$) ermittelt werden, so ist die Konstante zwischen -1 und 2 (z. B. mit einer Schrittweite von 0.2) zu verändern. Nachfolgende Bildschirmgrafik zeigt das mit der Beispieldatei BODE-DEMO.BSY erhaltene Ergebnis.



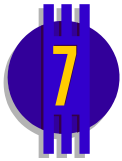
Frequenzgang aus BODE-DEMO.BSY

Während die Ermittlung der Betragskennlinie relativ unkritisch ist, sind für eine möglichst exakte Ermittlung der Phasenkennlinie die Einstellungen für Simulationsdauer und Schrittweite mit Bedacht zu wählen. Die Simulationsdauer sollte so groß gewählt werden, dass auch bei der kleinsten darzustellenden Frequenz noch mindestens zwei Schwingungen ermittelt werden. Umgekehrt muss die Simulationsschrittweite so klein gewählt werden, dass auch die größten darzustellenden Frequenzen noch hinreichend fein aufgelöst werden; zusätzlich spielen dabei natürlich auch die Kenngrößen des untersuchten Systems (Zeitkonstanten und Eigenfrequenzen) eine wesentliche Rolle. Bei ungünstiger Wahl von Simulationsdauer und/oder -schrittweite kann es zu Ungenauigkeiten in der Ermittlung des Phasengangs kommen.

Bei sehr zeitaufwendigen Frequenzgangberechnungen kann es sinnvoll sein, die Simulationsschrittweite mit Hilfe des VCOs automatisch zu steuern; dieser bietet dazu in seinem Parameterdialog eine entsprechende Option an und passt bei aktivierter Option die Simulationsschrittweite von BORIS automatisch an die aktuelle Frequenz an; hierdurch lässt sich insbesondere bei den niedrigen Frequenzen Rechenzeit einsparen, da diese dann mit wesentlich größerer Schrittweite simuliert werden. Weiterhin kann bei Bedarf über eine entsprechende Option im Parameterdialog des Frequenzgang-Plotters eine laufende Simulation automatisch abgebrochen werden, sobald die Ausgangsgröße eine genügende Zahl von Schwingungen absolviert hat; hierdurch lässt sich insbe-

sondere bei den hohen Frequenzen einige Rechenzeit einsparen. Beide Optionen sind jedoch mit Bedacht zu wählen!

Ermittlung von Frequenzgängen

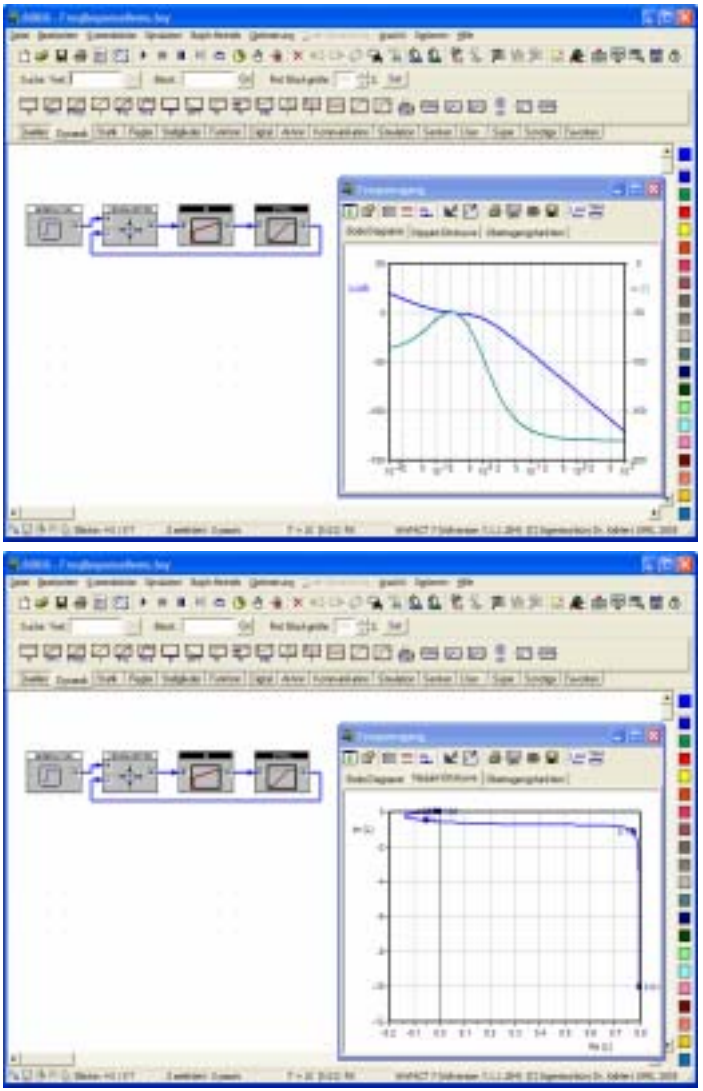


Neben der Berechnung von Zeitverläufen erlaubt BORIS auch die Ermittlung von *Frequenzgängen* linearer Systeme. Diese können in folgenden Formen dargestellt werden:

- Bode-Diagramm (Betrags- und Phasenkennlinie)
- Nyquist-Ortskurve (Real- und Imaginärteil)
- Übertragungsfunktion (Zähler- und Nennerpolynom)

Zur Ermittlung des Frequenzgangs einer Teilstruktur müssen zunächst die entsprechenden Blöcke selektiert werden. Anschließend erfolgt dann die Berechnung des Frequenzgangs, wobei – unabhängig von der tatsächlichen Verschaltung der Blöcke – immer von einer *Reihenschaltung* der Blöcke ausgegangen wird; d. h. beispielsweise, dass auch bei Selektion zweier parallelgeschalteter Blöcke dennoch der Frequenzgang der Reihenschaltung beider Blöke ermittelt wird. Weiterhin kann die Ermittlung des Frequenzgangs nur dann erfolgen, wenn ausschließlich Systemblöcke mit *linearem* Übertragungsverhalten selektiert wurden. Nach Anwahl der zu analysierenden Blöcke kann die Ermittlung des Frequenzgangs dann über die Menüoption BEARBEITEN | FREQUENZGANG ERMITTELN oder die Schaltfläche gestartet werden.

Die Ausgabe des Frequenzgangs erfolgt in einem separaten Fenster, welches eine eigene Palette für jede der drei oben angegebenen Darstellungsformen des Frequenzgangs besitzt. Nachfolgende Bildschirmgrafik zeigt als Beispiel Bode-Diagramm und Nyquist-Ortskurve eines offenen Regelkreises bestehend aus einem PI-Regler und einer PT1T2-Strecke.



Beispiel für Bode-Diagramm (oben) und Ortskurve (unten) eines offenen Regelkreises (Beispieldatei FreqResponseDemo.bsy*)*

Neben dem Frequenzgang des offenen Regelkreises (d. h. der Reihenschaltung aller selektierten Blöcke) kann auch der Frequenzgang des zugehörigen geschlossenen Regelkreises ermittelt werden; zwischen beiden Darstellungsformen wird über die Schaltflächen und des Frequenzgang-Fensters umge-

schaltet werden. Wurde eine neue Teilstruktur selektiert oder wurden Parameter einzelner Blöcke geändert, so muss der Inhalt des Frequenzgang-Fensters jeweils über die Schaltfläche aktualisiert werden. Über die Schaltfläche kann der dargestellte Frequenzbereich modifiziert werden. Daneben besitzt das Fenster eine Reihe weiterer Schaltflächen (z. B. zum Speichern oder Drucken von Ergebnissen), deren Funktion im Wesentlichen selbsterklärend ist.

Soll anstelle des grafischen Frequenzgangs lediglich eine Ermittlung der Übertragungsfunktion erfolgen, so kann dieses auch über die Menüoption BEARBEITEN | ÜBERTRAGUNGSFUNKTION ERMITTELN... geschehen.

Hinweis: Sofern der offene Regelkreis eine Totzeit aufweist, wird diese im Frequenzgang bzw. der Übertragungsfunktion des geschlossenen Regelkreises *nicht* berücksichtigt.

Numerische Optimierung von Systemparametern

BORIS bietet über die reine Simulation hinaus weitreichende Möglichkeiten zur numerischen Optimierung von Systemparametern mit Hilfe von leistungsfähigen Optimierungsverfahren, sogenannten *Evolutionsstrategien* [14]. Diese sind in der Lage, das gesuchte globale Optimum auch bei sehr zerklüfteten Topologien der Gütefunktion (Zielfunktion) mit einer hohen Zuverlässigkeit zu finden.

Zur Einführung in die numerische Optimierung mit BORIS soll zunächst ein rein mathematisches Optimierungsproblem betrachtet werden. Gesucht seien die Optimierungsparameter $x_1 \ldots x_5$, für die die Gütefunktion

$$Q(x_1,...,x_5) = (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2 + (x_4 - 4)^2 + (x_5 - 5)^2$$

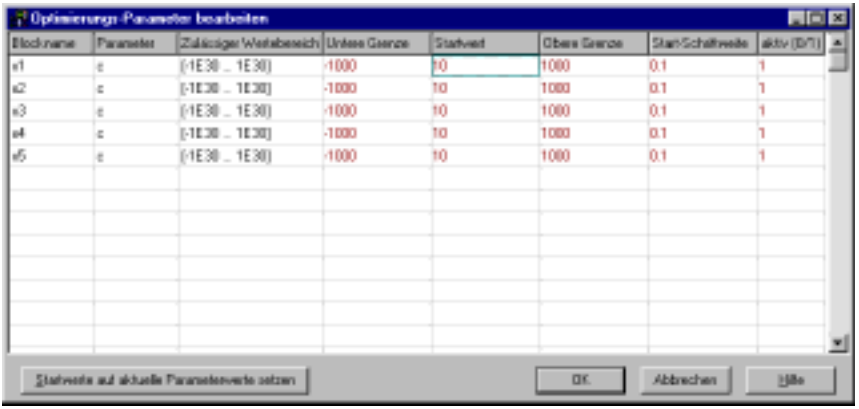
ihr Minimum annimmt. Es sei lediglich bekannt, dass alle Parameter im Bereich $-1000 \le x_i \le 1000$ liegen müssen. Anhand dieses Beispiels lässt sich die Qualität der gefundenen Lösung leicht überprüfen, da das exakte Optimum bekannt ist; es liegt bei

$$x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 4,\ x_5 = 5\ .$$

Der zugehörige Wert der Gütefunktion liegt bei null.

Festlegung der Optimierungsparameter

Optimiert werden können alle aktiven Exportparameter der aktuellen Systemebene. Da in obigem Beispiel fünf Parameter zu optimieren sind, wählt man entsprechend fünf Systemblöcke vom Typ *Konstante* und benennt diese mit $x_1 \ldots x_5$. Der Parameter *c* jedes Blocks wird dann als Exportparameter aktiviert. Zur Bearbeitung der Optimierungsparameter dient anschließend die Option OPTIMIERUNG | OPTIMIERUNGSPARAMETER....

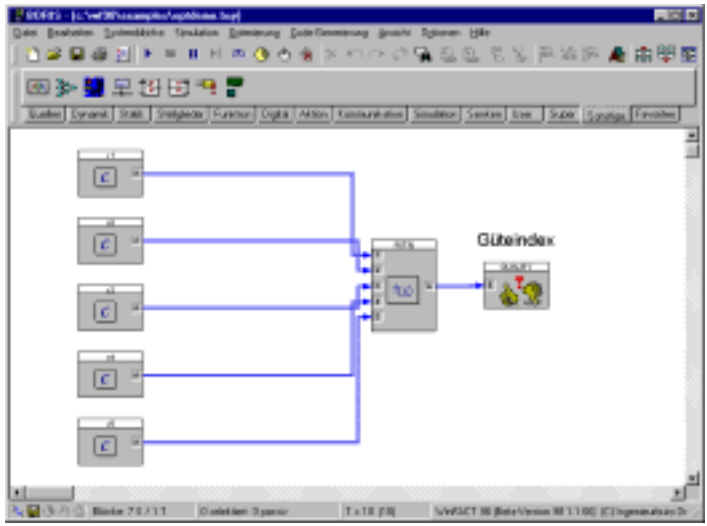


Dialog zur Bearbeitung der Optimierungsparameter

Der zugehörige Dialog erlaubt nun für jeden Optimierungsparameter die Vorgabe der unteren und oberen Grenze, innerhalb der die Optimierung durchgeführt werden soll (in obigem Beispiel -1000 bzw. 1000), des Startwertes für die Optimierung (oben 10) sowie einer Start-Schrittweite (oben 0.1 für alle Parameter). Außerdem können einzelne Parameter von der Optimierung ausgeschlossen werden, indem sie durch einen Eintrag 0 in der Spalte *aktiv (0/1)* deaktiviert werden. Dies ist z. B. sinnvoll, wenn bei einem Block mit drei Exportparametern nur zwei dieser Parameter auch optimiert werden sollen. Über die Schaltfläche *Startwerte auf aktuelle Parameter setzen* ist es möglich, im Anschluss an eine bereits durchgeführte Optimierung einen neuen Lauf zu starten, der mit den Ergebnissen der vorangegangenen Optimierung als Startwerten beginnt.

Definition des Gütekriteriums

BORIS versteht unter einer Optimierung grundsätzlich die *Minimierung* einer Güte- oder Zielfunktion, die innerhalb der zugrunde liegenden Systemstruktur einfach grafisch definiert werden kann. Dazu dient ein sogenannter *Güteindex-Block*, der über OPTIMIERUNG | GÜTEINDEX-BLOCK EINFÜGEN wie ein gewöhnlicher Systemblock eingefügt werden kann. Das in diesen Block einfließende Signal stellt anschließend die Gütefunktion dar. Untenstehende Grafik zeigt die Optimierungsstruktur für das Beispiel-Problem, das als OPTDEMO.BSY im Examples-Verzeichnis zu finden ist. Der Funktionsblock dient dabei zur Berechnung der eigentlichen Gütefunktion $Q(x_1,...,x_5)$.

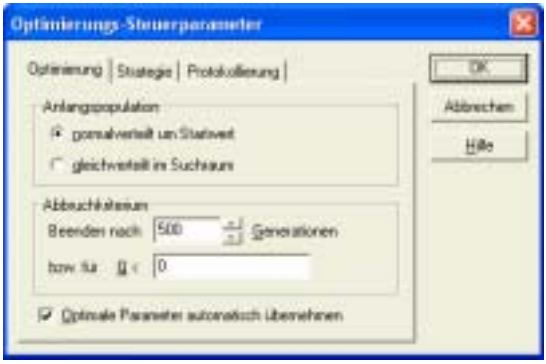


Optimierungsstruktur für Beispiel-Problem

Steuerparameter für die Optimierung

Das Optimierungsverfahren auf der Basis von Evolutionsstrategien (Genetischen Algorithmen) besitzt eine Reihe von Steuerparametern (Strategieparametern), über die die Optimierung beeinflusst werden kann. Diese sind über OPTIMIERUNG | STEUERPARAMETER... zugänglich. Der zugehörige Dialog ist in drei Palettenseiten aufgeteilt.

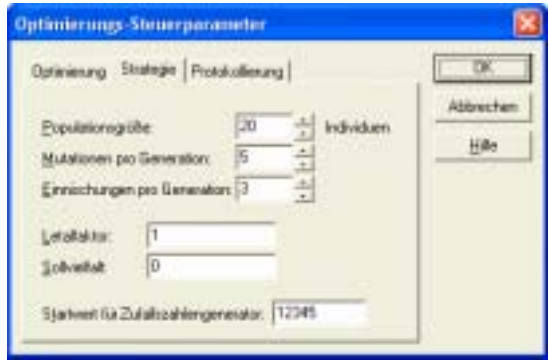
Palettenseite *Optimierung*



Diese Palettenseite umfasst folgende Optionen:

Anfangspopulation	Diese Einstellung legt die Art der Anfangspopulation fest. In der Einstellung *normalverteilt um Startwert* werden alle Individuen normalverteilt um den unter *Optimierungsparametern* eingestellten Startwert generiert. Die Streuung der Normalverteilung entspricht dabei der gewählten Anfangsschrittweite. In der Einstellung *gleichverteilt im Suchraum* wird die Anfangspopulation gleichtverteilt innerhalb der vorgegebenen Parametergrenzen erzeugt.
Abbruchkriterium	Legt das Abbruchkriterium für die Optimierung fest. Diese wird beendet, wenn eine vorgebbare Zahl von Generationen überschritten wurde oder die Gütefunktion einen bestimmten Grenzwert unterschreitet.
Optimale Parameter automatisch übernehmen	Ist diese Option aktiviert, werden nach Beendigung der Optimierung die ermittelten optimalen Parameter automatisch in die entsprechenden Blockparameter übernommen.

Palettenseite *Strategie*

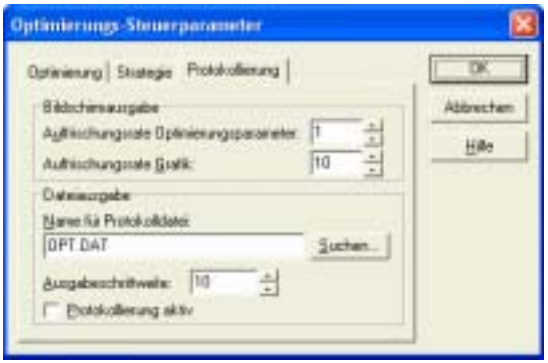


Diese Palettenseite umfasst folgende Optionen:

Populationsgröße	Anzahl der Individuen in der Population. Dieser Wert sollte an die Anzahl der Optimierungsparameter angepasst werden und mindestens vier bis fünf mal so groß sein.
Mutationen pro Generation	Legt die Anzahl der mutierten Individuen pro Generation fest. Ein höherer Wert führt zu einer mehr globalen Suche nach dem Optimum, kann aber zu einer verlangsamten Konvergenz führen (siehe dazu [14]).
Einnischungen pro Generation	Legt die Anzahl der Einnischungen pro Generation fest. Ein höherer Wert führt zu einer mehr globalen Suche nach dem Optimum, kann aber zu einer verlangsamten Konvergenz führen (siehe dazu [14]).
Letalfaktor	Legt den Letalfaktor für die Mutationen fest. Ein niedriger Wert führt zu einer mehr globalen Suche nach dem Optimum, kann aber zu einer verlangsamten Konvergenz führen (siehe dazu [14]).
Sollvielfalt	Legt die Sollvielfalt der Individuen innerhalb der Population fest. Ein höherer Wert führt zu einer mehr globalen Suche nach dem Optimum, kann aber zu einer verlangsamten Konvergenz führen (siehe dazu [14]).
Startwert für Zufallszahlengenerator	Legt den Startwert für den Zufallszahlengenerator fest, der u. a. zur Erzeugung von Mutationen benutzt

wird.

Palettenseite *Protokollierung*

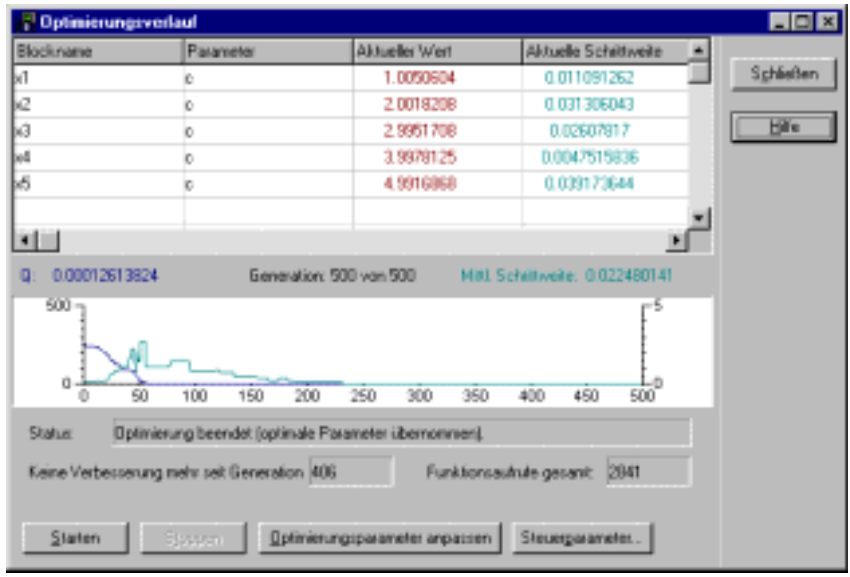


Diese Palettenseite umfasst folgende Optionen:

Bildschirmausgabe	Legt die Refreshraten für die Bildschirmausgabe fest (siehe Abschnitt *Steuern des Optimierungsablaufs*)
Dateiausgabe	Diese Einstellungen ermöglichen die Protokollierung des Optimierungsverlaufs in einer Textdatei. Der im Feld *Ausgabeschrittweite* angegebene Wert legt bei einer aktiven Protokolldatei fest, nach jeder wievielten Generation ein Abspeichern erfolgt.

Steuern des Optimierungsablaufs

Sind alle Vorbereitungen beendet, gelangt man über OPTIMIERUNG | OPTIMIERUNG STARTEN... in den Dialog zur Optimierungssteuerung. Nachstehende Bildschirmgrafik zeigt den Dialog nach einem erfolgreichen Optimierungslauf für das behandelte Beispiel-Problem.



Dialog zur Optimierungssteuerung (hier nach einer bereits beendeten Optimierung)

Dieser Dialog zeigt im oberen Teil alle Optimierungsparameter mit ihren aktuellen Werten (rot) und Schrittweiten (blaugrün) an. Die Grafik im mittleren Teil zeigt während der Optimierung den zeitlichen Verlauf der Gütefunktion und der mittleren Schrittweite an. Zur weiteren Information dient die Anzeige der letzten erfolgreichen Generation und der Gesamtzahl der Funktionsaufrufe, d. h. Simulationsläufe.

Ein neuer Optimierungslauf wird über die Schaltfläche *Starten* gestartet und kann dann jederzeit über die *Stoppen*-Schaltfläche abgebrochen werden. Nach Beendigung des Laufs können über *Optimierungsparameter anpassen* die aktuellen Optimierungsparameter als neue Startwerte und die aktuellen Schrittweiten als neue Start-Schrittweiten übernommen werden, um basierend auf diesen Einstellungen einen neuen Optimierungslauf zu starten.

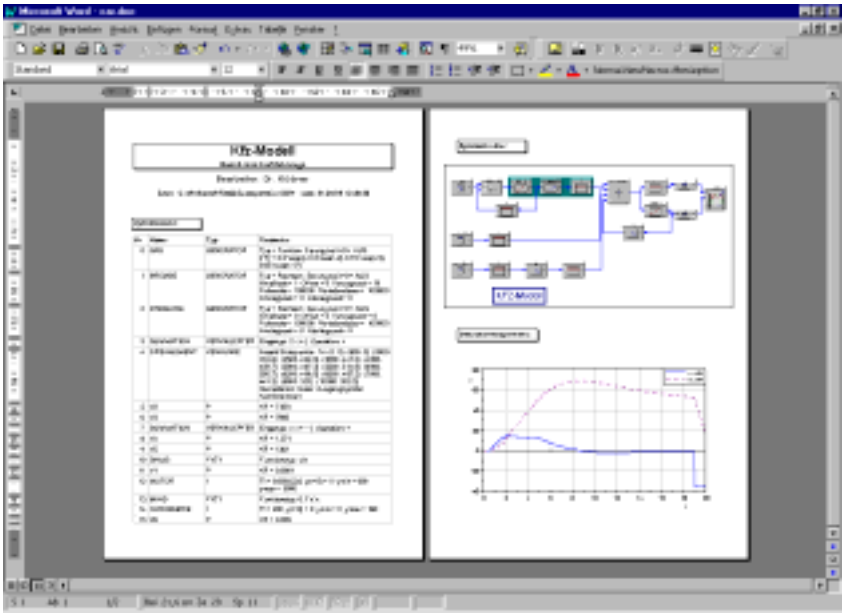
Anwendung zur Regleroptimierung

Eine wesentliche Anwendung der numerischen Parameteroptimierung liegt in der automatischen Optimierung von Reglerparametern, beispielsweise nach dem ITAE-Kriterium oder beliebigen Gütekriterien. Dabei kann auf einfache Weise z. B. auch der maximale Stellgrößenbedarf des Reglers im Gütekriterium berücksichtigt werden. Ebenso können mehrere Regler simultan, d. h. par-

allel optimiert werden. Zwei Anwendungsbeispiele hierzu finden Sie im Examples-Verzeichnis in den Dateien ITAEOPT.BSY und ITAEOPT2.BSY.

Dokumentation von Systemen

BORIS ermöglicht über seinen integrierten Dokumentgenerator die komfortable Dokumentation einer kompletten Systemstruktur sowohl in grafischer Form als auch in Textform. Dabei kann die Ausgabe sowohl direkt auf dem Drucker als auch in einer Vielzahl von Ausgabeformaten erfolgen, so dass eine Weiterverarbeitung mit anderen Standardanwendungen möglich ist.

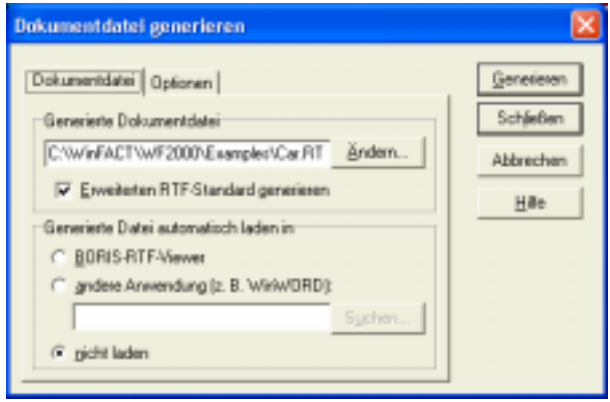


Dokumentation eines Systems mit dem integrierten BORIS-Dokumentgenerator

Erzeugung der Dokumentdatei

Sobald eine BORIS-Systemdatei bzw. Superblockdatei einmal gespeichert wurde, kann für diese Datei eine Dokumentdatei generiert werden. Dies ist eine Textdatei im RTF-Format (*Rich Text Format*), die von praktisch jedem Textverarbeitsprogramm gelesen und weiterverarbeitet werden kann. Steht ein solches Textverarbeitungprogramm nicht zur Verfügung, kann auch der integrierte RTF-Viewer benutzt werden. Dieser ist allerdings von seinem Leistungsumfang her gegenüber Standard-Textverarbeitungsprogrammen stark eingeschränkt.

Zur Generierung der Dokumentation wählen Sie DATEI | DOKUMENTDATEI GENERIEREN... bzw. die Schaltfläche der System-Toolbar. Es folgt ein in mehrere Paletten aufgeteilter Dialog, über den die Dokumentdatei-Generierung gesteuert werden kann.



Dialog zur Steuerung der Dokumentgenerierung (Palette Dokumentdatei*)*

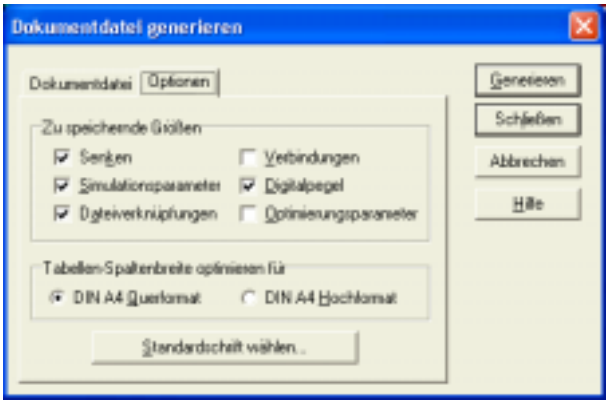
Im Eingabefeld *Generierte Dokumentdatei* kann der Name der generierten RTF-Datei angegeben werden. Dieser entspricht per Voreinstellung dem Namen der aktuellen Systemdatei, jedoch mit der Extension RTF. Ist das Optionsfeld *Erweiterten RTF-Standard generieren* ausgewählt, so wird die Datei in Tabellenform dargestellt, was zu einer wesentlich übersichtlicheren Darstellung führt. Der integrierte RTF-Viewer sowie einige einfache Texteditoren beherrschen diesen erweiterten Standard jedoch nicht, so dass die Option in diesen Fällen deaktiviert werden sollte.

Auf Wunsch kann die generierte Datei direkt in den entsprechenden RTF-Editor bzw. das Textverarbeitungsprogramm geladen werden. Dazu ist die

zugehörige Option im Gruppenfenster *Generierte Datei automatisch laden in* zu aktivieren.

Über die Palette *Optionen* des Dialogs können zunächst die in der Dokumentdatei abzuspeichernden Größen ausgewählt werden (Gruppenfenster *Zu speichernde Größen*). Im Gruppenfenster *Tabellen-Spaltenbreite optimieren für* wird das zugrundeliegende Papierformat für die Datei festgelegt. Über die Schaltfläche *Standardschrift wählen* kann schließlich der Schrifttyp für die Ausgabe festgelegt werden.

Die eigentliche Dokumentgenerierung wird über die Schaltfläche *Generieren* eingeleitet. Im Anschluss daran wird – sofern diese Option nicht zuvor deaktiviert wurde – die generierte Datei automatisch in die gewählte Textverarbeitung geladen.



Palette Optionen

Exportieren der Systemstruktur

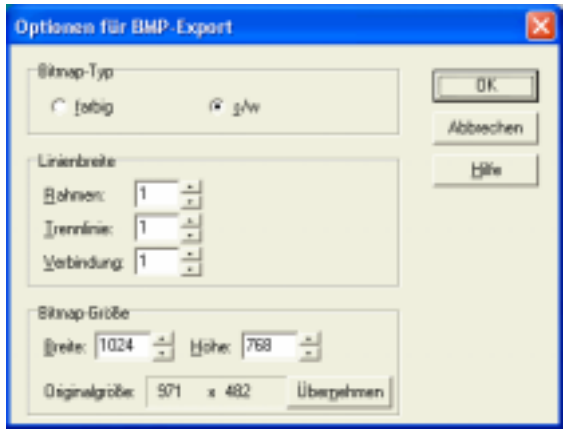
Die Systemstruktur kann wahlweise in zwei Formaten exportiert werden:

- Als Bitmap-Grafik im BMP-Format
- Als Vektorgrafik im WMF-Format

Beide Formate können sowohl farbig als auch im s/w-Format exportiert werden und sind praktisch von allen Windows-Anwendungen (Textverarbeitung, Grafikprogramm, Präsentationsprogramm usw.) weiterverarbeitbar. Zum Export dient die Menüoption DATEI | EXPORTIEREN.... Im Anschluss daran erscheint ein Windows-Dateiauswahldialog, über den das gewünschte Exportformat und

der Dateiname festgelegt werden können. Abhängig vom gewählten Format kann nachfolgend die Exportgrafik genauer spezifiziert werden.

Export-Optionen für WMF-Format

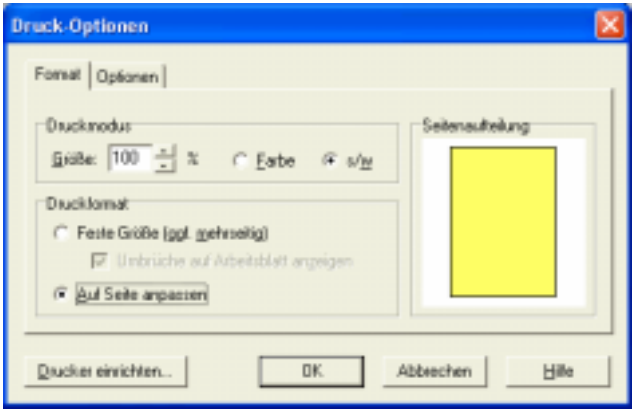


Export-Optionen für BMP-Format

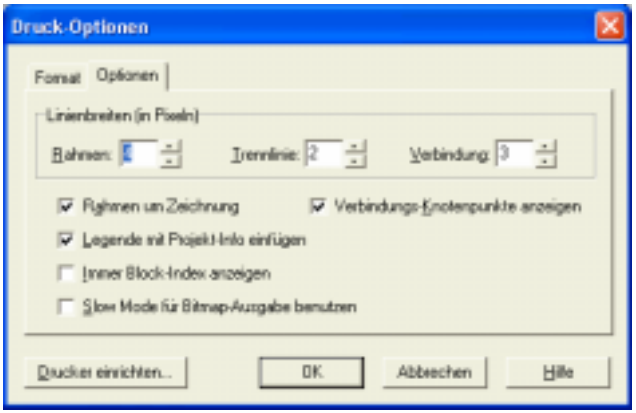
Drucken der Systemstruktur

Alternativ zum Export der Systemstruktur kann diese auch direkt aus BORIS heraus gedruckt werden. Dazu dient die Menüoption DATEI | DRUCKEN... bzw. die Schaltfläche der System-Toolbar. Der Ausdruck kann durch eine Reihe

von Optionen gesteuert werden, die über einen entsprechenden Dialog vorgegeben werden können.



Druck-Optionen-Dialog, Palette Format



Druck-Optionen-Dialog, Palette Optionen

Die einzelnen Optionen haben folgende Bedeutungen:

Option	**Bedeutung**
Druckmodus	Legt die Größe des Ausdrucks sowie die Farbauflösung fest
Druckformat	Legt fest, ob der Ausdruck mit einer festen Größe - ggfls. verteilt auf mehrere Seiten - erfolgen oder

	aber automatisch an die Seitengröße angepasst werden soll. Bei einer mehrseitigen Ausgabe können die Seitenumbrüche durch Aktivierung der Option *Seitenumbrüche auf Arbeitsblatt anzeigen* bereits zur Entwurfszeit kontrolliert werden. Die aktuelle Seitenaufteilung wird ferner im Feld *Seitenaufteilung* angezeigt.
Linienbreiten in Pixeln	Ermöglicht die Vorgabe der Linienbreiten für die verschiedenen Komponenten der Systemstruktur
Rahmen um Zeichnung	Legt fest, ob ein Rahmen um die Grafik gedruckt werden soll
Legende mit Projekt-Info einfügen	Ist diese Option aktiviert, enthält die Grafik automatisch eine Legende mit den wichtigsten Daten der zugehörigen Projekt-Information.
Immer Block-Index anzeigen	Ist diese Option aktiviert, wird unabhängig von der Einstellung für die Bildschirmausgabe beim Drukken grundsätzlich der Block-Index angezeigt.