

WinBasic

Inhaltsverzeichnis

- 1 Vorbemerkungen
- 2 Handhabung
- 3 Einführung
- 4 Referenzteil

Anhang A Runtime-Errors

1 Vorbemerkungen

Als Ende 1989 FranzBasic auf den Markt kam, hatten wir angekündigt, das wir diesen Basic-Übersetzer auch für andere Betriebssysteme als MS-DOS anbieten wollten. Dabei wurde hauptsächlich an UNIX und OS/2 gedacht.

Inzwischen ist die grafische Benutzeroberfläche Microsoft Windows in ihrer Beliebtheit ständig gestiegen. Was fehlt sind die Fülle von Anwendungs-Programmen wie MS-DOS sie bietet und - damit eng verbunden - eine einfache und leistungsfähige Möglichkeit zur Entwicklung von Programmen für MS-Windows. Während für MS-DOS eine Vielzahl von verschiedenen Programmiersprachen für verschiedene Anwendungsgebiete existieren, kommt für die Entwicklung von Windows-Programmen in der Praxis nur die Arbeit mit 'C' und dem Software Development Kit von Microsoft in Betracht.

Selbst für den professionellen Software-Entwickler ist dieser Weg hart und steinig, die Lernphase lang und kostspielig. Dem Software-Entwickler, der sich weniger als Programmier-Profi den als Problemlöser sieht und der unter MS-DOS gerne mit Programmiersprachen wie Turbo-Pascal oder mit Basic arbeitet, bleibt die Windows-Welt verschlossen.

WinBasic wird hier eine Lücke füllen. Es ist preiswert und einfach in der Anwendung, wie man es der Sprache Basic nachsagt. Durch die leistungsfähigen Funktionen auf hohem Niveau (verglichen mit den 450 low-level-Funktionen des Software Development Kit von Microsoft) wird die Programmierung von Windows zum Vergnügen. Die Schnelligkeit der Übersetzung, das Wegfallen eines Linker-Laufs und die ständige Verfügbarkeit eines integrierten Editors tun ein übriges.

Wir hoffen, das Sie viel Freude mit WinBasic haben werden. Sollten sich Probleme einstellen, senden Sie uns bitte Ihr Programm auf einer Diskette und mit einer Problembeschreibung zu.

WinBasic

Copyright Zimmer Informatik, Postfach 140540, Düsseldorf

Verwendete Warenzeichen:

Turbo Pascal	Borland
Dbase III	Ashton-Tate
MS-DOS	Microsoft
Microsoft Windows	Microsoft

Weitere Information

Auf der Diskette finden Sie eine Datei 'Liesmich', welche Informationen enthält, die zur Zeit der Drucklegung dieses Handbuchs noch nicht bekannt waren. Bitte sehen Sie sich die Information in dieser Datei an.

Installation

Zur Installation legen Sie bitte die Auslieferungsdiskette in das Diskettenlaufwerk 'a' und tippen den Befehl:

```
a:setup
```

ein. Natürlich können Sie auch andere Laufwerke benutzen.

Das Setup-Programm erstellt auf Ihrer Festplatte 'c' das Verzeichnis 'winbasic' und kopiert die Dateien auf der Diskette in dieses Verzeichnis.

2 Handhabung

2.1 Übersetzer

Kern des Übersetzers ist das Programm 'Winbasic.exe'. Sie starten den Übersetzer entweder durch Anklicken des Dateinamens 'Winbasic.exe' mit der Maus im Dateimanager von Windows oder indem Sie Windows mit dem Kommando

```
win winbasic [1]
```

starten. In letzterem Fall wird der Dateimanager nach dem Start von Windows als Symbol dargestellt.

Eine weitere Möglichkeit besteht darin, in der Datei 'win.ini' in der Sektion [extensions] folgende Zeile einzugeben:

```
bas=winbasic.exe ^.bas
```

Dann können Sie den Übersetzer durch Anklicken einer Datei mit der Extension '*.bas' starten. Windows lädt dann das Programm 'Winbasic.exe' und übergibt diesem den Namen der angeklickten Datei '*.bas'. Vorteil: die Datei wird von 'Winbasic.exe' unverzüglich in den Editor geladen. Analog zu oben gesagtem [1] können Sie auch beim Start von Windows eingeben:

```
win *.bas
```

und WinBasic wird mit der Datei '*.bas' im Editor beim Start von Windows aktiviert.

Nach dem Start erscheint das Fenster des Editors geöffnet auf dem Bildschirm, das Fenster des WinBasic-Programms '*.bas' ist als Symbol am unteren Bildschirmrand dargestellt. Beim Starten des Programms durch den Befehl 'Run' des Editor-Menüs wird letzteres automatisch geöffnet und beim Programmende wieder zum Symbol verkleinert.

Existiert für das WinBasic-Programm '*.bas' kein Symbol '*.ico' wird einfach ein weißes Rechteck gezeigt. Über das Erstellen von Symbolen für Programme in WinBasic lesen Sie bitte Absatz 2.3.

Anwender von Windows 3.0 haben auch die Möglichkeit, durch den Programm-Manager 'Winbasic.exe' in eine Programmgruppe aufzunehmen, eventuell mit dem Namen eines zu ladenden Programmtexts als Parameter. Lesen Sie hierzu in Ihrem Windows-Handbuch nach.

2.2 Das Runtime-Modul

Zu WinBasic gehört eine sogenannte 'Runtime-Modul', nämlich 'Runwinb.exe'. Dieses Modul bringt den Code ihres Programmes zur Ausführung, den Sie mit dem Menüpunkt 'Code' des Editors erzeugt haben und der vom Übersetzer in der Datei '*.cod' abgelegt wird. 'Runwinb.exe' können Sie zusammen mit ihren übersetzten Programmen '*.cod' unbeschränkt weitergeben, um anderen die Nutzung Ihrer Programme ohne Weitergabe

Ihres Programmtextes zu ermöglichen. Mit 'Runwinb.exe' müssen die Dateien im Verzeichnis '\zwin', welches zum Lieferumfang von WinBasic gehört, weitergegeben werden.

Ihr Programm-Code wird durch Anklicken der Code-Datei '*.cod' vom Runtime-Modul 'Runwinb.exe' ausgeführt. Alternativ können Sie bereits beim Start von Windows durch die Eingabe von

`win *.cod`

das Runtime-Modul aktivieren um ihr Programm '*.cod' zum Ablauf zu bringen.

Auf jeden Fall muß ähnlich wie für 'Winbasic.exe' in 'win.ini' unter der Sektion [extensions] die Zeile

`cod=runwinb.exe ^.cod`

eingetragen sein.

Für Anwender von Windows 3.0 gilt der letzte Absatz von Paragraph 2.1 entsprechend.

2.3 Hilfsprogramme

Fonts.exe

Dieses Hilfsprogramm erlaubt Ihnen festzustellen, welche Fonts Ihre Windows-Installation bereithält. Sie können über ein Menu zwischen Bildschirm-Fonts ('Screen') und Drucker-Fonts ('Printer') wählen und dann die Font-Familie anwählen, die Sie sich anschauen wollen. 'Fonts.exe' stellt dann alle verfügbaren Größen dieser Familie auf dem Bildschirm dar. In der Zeichenkette finden Sie eine Angabe ^h >w, dabei bedeutet h die Höhe und w die Weite des einzelnen Fonts in Punkten. Mit 'Clear' wird der Bildschirm gelöscht.

Wenn Sie etwas auf Bildschirm und Drucker ausgeben wollen, sollten Sie für die Programmierung die Drucker-Fonts auswählen, da die Darstellung auf dem Bildschirm i. a. nicht so kritisch ist wie die Druckausgabe.

Weitere Informationen zu Fonts finden Sie im Referenzteil, Anweisung FONT.

Bitmap.exe

Mit 'Bitmap.exe' können Sie grafische Daten aus dem Clipboard in eine Datei kopieren.

'Bitmap.exe' ist gedacht zur Herstellung von Symbolen für Ihre Basic-Programme. Sie können mit dem Windows-Programm 'Paint.exe' Zeichnungen in passender Größe erstellen wobei Ihnen die mitgelieferte Zeichnung 'Icon.msp' als Größenvorlage dienen kann. Selektieren Sie dann mit der Maus diesen Teil moeglichst exakt und bringen durch den Menubefehl 'Edit/Kopieren' die Zeichnung in das Clipboard. Durch das Aktivieren von 'Bitmap.exe' wird der Clipboard-Inhalt in eine Datei mit dem Namen 'Icon.ico' kopiert. Wenn Sie dann 'Icon.ico' passen zu Ihrem Programm umbenennen (z. B. 'Myprog.ico' für das Programm 'Myprog.bas') wird diese Zeichnung von Ihrem Programm als Symbol

verwendet.

'Bitmap.exe' öffnet kein Fenster.

2.4 Der Editor

Ihre Schnittstelle zum Übersetzer ist der Editor, mit dem Sie Ihre Programmtexte bearbeiten und mit dem Sie Kommandos an den Übersetzer geben. Das Fenster des Editor besteht aus verschiedenen Bereichen:

- dem Arbeitsbereich, in dem der Programmtext bearbeitet wird
- der Menuleiste, wie sie die meisten Windows-Programme haben
- der Statuszeile am unteren Ende des Editor-Fensters. Hier werden nach einem Übersetzungslauf etwaige Syntax-Fehler angezeigt, ansonsten wird das aktuelle Verzeichnis angezeigt.

In der Menuzeile des Editor sehen Sie fünf Menüpunkte, die beim Anklicken Abrollmenüs aktivieren. Die ersten vier Abrollmenüs enthalten Kommandos, die das Bearbeiten eines Programmtexts betreffen (Datei, Edit, Suchen, Optionen), während das fünfte (Run) Kommandos enthält, die das Übersetzen und Starten des Programms betreffen.

Die Menüpunkte können durch Anklicken mit der Maus oder durch Eingabe der unterstrichenen Buchstaben bei gleichzeitig gedrückter ALT-Taste aktiviert werden.

Sie sehen im Arbeitsbereich des Editors zwei verschiedene Cursors, den Maus-Cursor, welcher den Bewegungen der Maus folgt und einen strichförmigen Text-Cursor, der die aktuelle Schreibposition innerhalb des Programmtextes markiert. Der Maus-Cursor nimmt verschiedene Formen an. Innerhalb des Textbereichs ist er dem Text-Cursor ähnlich, hat jedoch halbkreisförmige Enden, im Menübereich und in der Statuszeile nimmt er die bekannte Pfeilform an. Wird innerhalb des Textbereichs die linke Maus-Taste gedrückt, so wird der Text-Cursor an die momentane Position des Maus-Cursors gesetzt.

Werden Tasten auf der Tastatur gedrückt, werden druckbare Zeichen an der aktuellen Position des Text-Cursors in den vorhandenen Text eingefügt. Einige Tasten haben Steuerfunktionen:

ALT	Menu aktivieren
Pfeiltasten	Text-Cursor bewegen
Tab	Tabulator
PgUp/Bild auf	eine Bildschirmseite aufwärts blättern
PgDn/Bild ab	eine Bildschirmseite abwärts blättern
Home/Pos1	Zeilenanfang
End/Ende	Zeilenende
Enter	Neue Zeile
Backspace	Zeichen links von Text-Cursor löschen
Del/Entf	Zeichen rechts von Text-Cursor löschen
Ins/Einf	zwischen 'Einfüge'- und 'Überschreib'-Modus wechseln
Ctrl-Ins	selektierten Text ins Clipboard kopieren
Shift-Del	selektierten Text löschen und in das Clipboard

	kopieren
Shift-Ins	Text aus dem Clipboard an der Position des Text-Cursors einfügen

Text wird selektiert, indem bei gedrückter 'Shift'-Taste der Text-Cursor mit den Pfeiltasten über den zu selektierenden Text bewegt wird. Der selektierte Text wird invers dargestellt. Eine andere Möglichkeit Text zu selektieren ist, den Maus-Cursor bei gedrückter linker Maus-Taste über den zu selektierenden Text zu bewegen. Ist ein Textstück selektiert, kann es den Befehlen aus dem Abrollmenu 'Edit' oder den äquivalenten Tastatur-Eingaben bearbeitet werden. Der komplette Text kann mit dem Befehl 'Alles selektieren' aus dem Abrollmenu 'Edit' selektiert werden.

Das 'Clipboard' ist eine Zwischenablage in Windows, in welcher der Editor bei bestimmten Befehlen Textstücke zur späteren Verwendung ablegt. Wird z. B. mit 'Shift-Del' selektierter Text gelöscht, so wird er im Clipboard abgelegt um eventuell mit 'Shift-Ins' an anderer Stelle wieder eingefügt werden zu können.

Ein Blättern durch den Text ist auch mit dem Rollbalken am rechten Fensterrand möglich.

Mit dem Drücken der Eingabe-Taste wird eine neue Zeile unterhalb der aktuellen eingefügt und auf diese gesprungen. Steht der Cursor dabei nicht hinter dem letzten Zeichen der alten Zeile wird diese geteilt.

Zwei Zeilen können vereinigt werden, indem man die 'Del'-Taste drückt, während der Cursor genau hinter dem letzten Zeichen der oberen Zeile steht

Die Menu-Befehle

Menu-Befehle können durch Anklicken mit der Maus oder durch Eingabe der unterstrichenen Buchstaben in den Befehlsworten bei gleichzeitig gedrückter ALT-Taste aktiviert werden.

Datei

Unter dem Abrollmenu 'Datei' finden sich im wesentlichen Befehle, die mit dem Laden und Speichern von Programmtexten und dem Beenden des Programmlaufs zu tun haben.

Mit 'Öffnen' wird eine neue Datei zur Bearbeitung als Programmtext geöffnet und in den Editor geladen. Ist bereits ein Programmtext geladen und wurde dieser geändert, wird die Sicherung des Textes in eine Datei angeboten. Der Name der zu bearbeitenden Datei kann über die Tastatur eingegeben oder aus einer 'Listbox' ausgewählt werden.

'Neu' löscht den Text im Editor. Wurde dieser Text geändert, wird die Sicherung des Textes in eine Datei angeboten.

'Sichern' sichert den geladenen Programmtext in die Datei aus der er geladen wurde. Bei Programmtext der nicht aus einer Datei geladen sondern neu erzeugt wurde, wird jedoch wie in 'Sichern als' verfahren.

'Sichern als' ermöglicht das Sichern des geladenen Programmtext in eine Datei mit

Dokumentation WinBasic8
beliebigen Namen.

'Drucken' gibt den Programmtext über den Drucker aus.

'Exit' beendet die Sitzung mit dem Editor und damit auch mit WinBasic

'Info' zeigt einem Copyright-Vermerk.

Edit

Die Menu-Befehle unter Edit dienen zur Veränderung des im Editor geladenen Texts. Sie wirken - bis auf 'Rücknahme' - auf selektierten Text.

'Rücknahme' nimmt die zuletzt gemachte Veränderung - wenn möglich - zurück.

'Löschen' löscht den selektierten Text. Keine Übernahme ins Clipboard.

'Kopieren' kopiert den selektierten Text ins Clipboard. Keine Veränderung des Programmtexts.

'Schneiden' kopiert den selektierten Text ins Clipboard und löscht ihn aus dem Programmtext.

'Einfügen' fügt Text aus dem Clipboard an die aktuelle Position des Text-Cursors ein.

'Alles selektieren' selektiert den gesamten Programmtext.

Suchen

'Finden' ermöglicht das Auffinden eines bestimmten Textstücks im Programmtext. Dazu wird eine Dialogbox in den Textbereich eingeblendet, um die Eingabe des Suchtextes zu ermöglichen. Mit der Option 'Ohne Fall' kann die Unterscheidung zwischen Groß- und Kleinschreibung aus- und eingeschaltet werden. Wird die Option 'Ersetzen durch' angewählt, kann ein Ersatztext eingegeben werden, der den gefundenen Suchtext ersetzt. Gesucht wird zunächst von der aktuellen Position des Text-Cursors bis zum Dateiende und danach vom Dateianfang zur aktuellen Position.

'Finde Nächsten' wiederholt die Aktion von 'Finden', ohne durch die Dialogbox eine erneute Eingabe zu verlangen.

Optionen

Mit 'Einfügemodus' kann zwischen dem Einfügen zwischen vorhandenen Zeichen und dem Überschreiben vorhandener Zeichen bei der Eingabe von Programmtext hin- und hergeschaltet werden.

Mit 'Autosave' kann die automatische Datensicherung ein- und ausgeschaltet werden. Die automatische Datensicherung speichert alle 500 Tastendrücke den Programmtext "*.bas" in eine Datei "*.bkp". Während des Speichervorgangs wird der Uhrglas-Cursor gezeigt.

Die Option 'Debug' zeigt dem Übersetzer an, daß außer dem Programmcode

Informationen zum Austesten von Programmen erstellt werden müssen. Ein mit 'Debug' übersetztes Programm überwacht Bereichsüberschreitungen von Feldindizes und ermöglicht die Ausführung von TRACE-Anweisungen (siehe Referenzteil).

Wegen der zusätzlich erzeugten Informationen wird das übersetzte Programm größer und etwas langsamer. Man sollte die Option 'Debug' sofort wieder ausschalten, wenn sie nicht mehr benötigt wird.

Die Optionen 'Einfügemodus' und 'Autosave' sind bei Programmstart eingeschaltet, 'Debug' ist ausgeschaltet. Im eingeschalteten Zustand sind die Menueinträge mit einem Haken versehen.

Run

Hier finden sich alle Befehle die mit der Programm-Übersetzung und -Ausführung zu tun haben. Für die Befehle wurden englische Begriffe gewählt, da diese inzwischen international gebräuchlich sind.

'Start' ist eine Kombination der Befehle 'Compile' und 'Run'.

'Compile' übersetzt den geladenen Programmtext. Ist die Übersetzung nicht ohne Fehler abgelaufen, wird der erste erkannte Fehler in der Statuszeile eingeblendet, dabei wird der Text-Cursor auf die entsprechende Stelle im Programmtext gesetzt.

'Run' führt ein Programm aus, wenn es fehlerfrei übersetzt wurde. Das Hauptfenster des Programms wird dabei - wenn es bislang als Symbol dargestellt wurde - auf normale Größe gebracht, das Fenster des Editors geht in den Hintergrund. Hat das auszuführende Programm kein 'Ende'-Befehl im Menu, wird es durch Anwahl des Kommandos 'Schließen/Close' in seinem System-Menu beendet. Es wird dabei zum Symbol verkleinert und der Editor tritt wieder in den Vordergrund.

'Error' zeigt bei nicht fehlerfrei übersetzten Programmen den jeweils nächsten Fehler an, dabei wird der Text-Cursor auf die Fehlerstelle positioniert. Eine einfachere Möglichkeit, den nächsten Fehler anzuzeigen ist, einfach den Maus-Cursor auf die Statuszeile zu positionieren und die linke Maus-Taste zu drücken. Nach Anzeige des letzten Fehlers wird wieder das aktuelle Verzeichnis in der Statuszeile angezeigt, bei der nächsten Anforderung wird wieder mit dem ersten Fehler begonnen. Die Anzahl der Fehler in der Liste ist auf 20 beschränkt.

'Code' generiert - falls das Programm fehlerfrei ist - einen Programmcode in eine Datei mit der Erweiterung '*.cod'. Diese Datei kann dann durch das Programm 'Runwinb.exe' abgearbeitet werden, ohne daß der Programmtext benötigt wird. Dies erlaubt eine Weitergabe des übersetzten Programms an Benutzer die nicht über WinBasic verfügen oder falls der Programmtext geschützt werden soll.

Hilfe

Dieser Befehl bietet Ihnen eine Online-Hilfe zu den Befehlen von WinBasic. Selektieren Sie im Programmtext den Befehl, zu dem Sie Hilfe wünschen, und wählen Sie dann den Menüpunkt 'Hilfe' an. Auf dem Bildschirm erscheint dann in einem Fenster eine

Dokumentation WinBasic10

Beschreibung der Syntax und der Befehlsparameter mit ihren möglichen Werten.

3 Einführung in WinBasic

Inhalt

3.1	Programmaufbau in WinBasic
3.2	Variablen in WinBasic
3.3	Konstanten
3.4	Arithmetische Ausdrücke
3.5	String-Ausdrücke
3.6	Logische Ausdrücke
3.7	Datei-Bearbeitung
3.8	Zeichensätze
3.9	Beispielprogramme

3.1 Programmaufbau in WinBasic

Programme bestehen aus einer Ansammlung von *Programmanweisungen*, auch einfach 'Anweisungen' oder 'Statements' genannt.

Mehrere Statements oder Programmanweisungen können auf einer Zeile stehen. Sie werden dann durch das Zeichen ':' getrennt. Es gibt einige Ausnahmen:

- nach einzeliligen If-Anweisungen wie

If a% < 0 Then a% = -a%

können keine weitere Anweisungen gegeben werden. Es muß dann die mehrzeilige Form gewählt werden (siehe 'IF').

- nach Sprungmarken können keine Anweisungen folgen
- nach den 'strukturierten' Anweisungen sind keine weiteren Anweisungen möglich

Programmanweisungen sind unterschieden in zwei Klassen: ausführbare und nicht ausführbare Anweisungen. Ausführbare Anweisungen werden beim Ablauf des Programms ausgeführt und führen irgendeine Änderung in Daten oder im Programmzustand herbei. Nichtausführbare Anweisung dienen zur Definition und Deklaration und werden bei der Übersetzung ausgewertet, z. B. DIM, RECTYPE, STATIC und andere.

Programme können strukturiert werden durch Anweisungen, die innerhalb eines Programmes Blöcke von Programmanweisungen bilden:

- Subroutine ... Endsub
- For ... Endfor
- If ... Endif
- While ... Wend
- Repeat ... Until

sind solche Anweisungspaare, die alle Statments '...' zwischen der einleitenden

(Subroutine, For, If, While, Repeat) und der abschliessenden Anweisung (Endsub, Endfor, Endif, Wend, Until) zu einem Programmblock schliessen. Diese Anweisungen werden oft auch strukturierte Anweisungen genannt.

Nach diesen strukturierten Anweisungen darf ebenfalls keine durch ':' abgetrenntes weiteres Statement stehen, wohl aber dürfen die Zeilen innerhalb des Blocks mehrere Anweisungen haben.

Die in einem Programmtext vorkommenden Wörter gehören in zwei Klassen, die *reservierten Wörter* und die vom Programmierer definierten Wörter. Reservierte Wörter sind diejenigen, die den Sprachumfang ausmachen, also 'IF', 'SUBROUTINE', 'GOTO', einschließlich der eingebauten Funktionen wie SIN(), TRIM\$(). Vom Programmierer werden dazu Variablen und Konstanten definiert. Variablen sollten nicht die Namen reservierter Wörter benutzen.

Zwischen Klein- und Großbuchstaben wird weder bei reservierten Wörtern noch bei Variablen unterschieden, wohl aber innerhalb von Zeichenketten. Die Variable ABC\$ kann Abc\$, abc\$, aBc\$ geschrieben werden. Für die in ABC\$ gespeicherte Zeichenkette gilt jedoch nicht, daß "Zeichenkette" gleich "zeichenKette" ist.

Ein *Programmtext* besteht aus Zeilen mit Programmanweisungen, die sich wiederum aus reservierten Wörtern und vom Programmierer definierten Wörtern zusammensetzen. Der Übersetzer macht aus diesem Programmtext einen *Programmcode*, der durch ein Runtime-Modul ausgeführt wird.

Ein wichtiges Element moderner Programmiersprachen sind Unterprogramme, in WinBasic realisiert durch das Anweisungspaar 'SUBROUTINE' ... 'ENDSUB'. Unterprogramme sind in sich abgeschlossene Programmteile, die wiederholt innerhalb eines Programmes verwendet werden können aber nur einmal programmiert werden müssen und auch nur einmal Platz für Programmtext und Programmcode beanspruchen.

Unterprogramme stehen in WinBasic vorzugsweise am Anfang eines Programmtexts, da sie definiert sein müssen, bevor sie aufgerufen werden können. Nach dem ENDSUB des letzten Unterprogramms beginnt automatisch das Hauptprogramm. Durch die DECLARE-Anweisung kann ein Unterprogramm deklariert werden, wenn es einmal aufgerufen wird (durch ein anderes Unterprogramm) bevor es definiert wird; trotzdem bleibt das Hauptprogramm immer das letzte Teilstück des Programmtexts.

Aufbau des Hauptprogramms

Programme werden im wesentlichen von Vorne beginnend abgearbeitet und enden - wenn einmal von Sprüngen, Schleifen usw. abgesehen wird - mit der letzten Zeile. Alle Aktionen des Programm-Benutzers werden vom Programm kontrolliert, z. B. kann der Benutzer eine Auswahl in einem Befehlsmenu nur machen, wenn das Programm ihm eine Menu anbietet, er kann eine Daten-Eingabe nur machen, wenn das Programm ihn dazu auffordert.

Mit Programm-Benutzer ist derjenige gemeint, der das fertige WinBasic-Programm nach Fertigstellung anwendet. Er ist zu unterscheiden vom Programmierer, welcher mit

WinBasic das Programm erstellt. Beide Personen können natürlich identisch sein. Unter Windows ist das anders. Der Benutzer kann in jedem Moment des Programmablaufs ein Menüitem anklicken, er kann zwischen verschiedenen Programmen hin- und herwechseln, er kann jederzeit mit der Maus arbeiten. Daher ist die Struktur eines Programms unter Windows wesentlich verschieden von einem Programm, das unter MS-DOS läuft.

Das Hauptprogramm eines mit WinBasic erstellten Programms ist aus der Sicht von Windows ein Unterprogramm, das wann immer es nötig ist aufgerufen wird. Das ist der Fall wenn der Benutzer einen Menubefehl auswählt, eine Taste auf der Tastatur drückt oder eine Taste auf der Maus betätigt. Das Hauptprogramm ist dann für die Reaktion auf diese Aktivität des Benutzers verantwortlich.

Soll das Hauptprogramm auf eine Aktion des Benutzers reagieren, schickt Windows dem Programm eine 'Message'. Die Message ist ein Code, in dem die Art der Benutzer-Aktion verschlüsselt ist. Das Hauptprogramm hält nun für alle Messages auf die es reagieren will eine Sprungmarke (Label) als Programm-Eintrittspunkt bereit. An diese Sprungmarke läßt Windows das Programm starten, wenn die entsprechende Message vorliegt.

Beispiel: Ein Hauptprogramm, das auf Tastatur- und Maus-Eingaben reagiert:

```
Rem *** Hauptprogramm
Print "Test MOUSE- und KEY-Message"
_MOUSE:
Mouse Knopf%, x%, y%
MoveTo x%, y%
Print "Knopf% ";Knopf%
_KEY:
Print inkey$()
```

Das Programm besteht aus einem Initialisierungsteil (Print "Test ...") und zwei Sprungmarken, die als Eintrittspunkte für Windows bei Messages dienen:

_MOUSE wird als Eintrittspunkt angesprungen, wenn der Programm-Benutzer auf eine Maustaste drückt

_KEY wird angesprungen, wenn der Benutzer auf eine Taste der Tastatur drückt, vorausgesetzt, es ist ein druckbares Zeichen oder einer der Funktionstasten F1 bis F9 und Shift-F1 bis Shift-F9

Wie alle Windows-Sprungmarken beginnen sie mit '_', um sie von gewöhnlichen Sprungmarken zu unterscheiden. Sprungmarken, die nicht mit dem Unterstrich beginnen sind keine Eintrittspunkte für Windows.

Beim Start des Programms läßt Windows den Initialisierungsteil ausführen, das ist der Teil eines Hauptprogramms, der vor der ersten Windows-Sprungmarke steht. Sobald diese Befehle vor der ersten Windows-Sprungmarke abgearbeitet sind (hier nur der eine Befehl PRINT) wird das Programm angehalten. Es ist nicht beendet, sondern 'suspendiert' und läuft an einer seiner Windows-Sprungmarken weiter, sobald Windows eine entsprechende Message schickt.

Drückt der Benutzer eine Taste auf der Tastatur, wird die Message 'KEY' gesandt. Das

Programm startet nun an der Sprungmarke `_KEY` und arbeitet die der Marke folgenden Befehle ab (hier ein `PRINT` Befehl), bis eine andere Windows-Sprungmarke (nicht gewöhnliche Sprungmarke!) oder das Programmende (im Sinne des Programmtext-Endes) erreicht. Das Programm wird nun wieder suspendiert und wartet auf einen weiteren Start durch Windows.

Wird eine der Maustasten betätigt, wird die Message 'MOUSE' generiert und das Programm am Eintrittspunkt `_MOUSE` wieder gestartet. Durch die der Sprungmarke folgenden Befehle holt sich das Beispielprogramm Informationen über den gedrückten Knopf und die Koordinaten der Maus beim Drücken dieses Knopfes und zeigt diese Information, in dem es auf dieser Stelle des Bildschirms die Nummer des gedrückten Knopfs schreibt. Da die Windows-Sprungmarke `'_KEY'` als nächste Anweisung folgt, wird das Programm wieder suspendiert.

Das Programm wird erst beendet, wenn der Benutzer den Befehl 'Schließen/Close' des Systemmenüs betätigt.

Was ist wenn neben dem immer möglichen Initialisierungsteil eines Programms auch ein Abschlussteil erforderlich ist, z. B. weil während des Programmlaufs andere Programme gestartet wurden, die wieder beendet werden müssen? (Dies ist unbedingt erforderlich, siehe 'START' im Referenzteil). Dann muß eine Menu her, das einen Befehl 'Ende' enthält.

Beispiel:

```
Dim Menu$(2)
```

```
Subroutine ProgEnde()
```

```
...
```

```
Endsub
```

```
Rem *** Hauptprogramm ***
```

```
Menu$(1)="Demomenu Ende Demo"
```

```
Menu$(2)=""
```

```
Menu Menu$
```

```
_1001:
```

```
  Gosub ProgEnde()
```

```
  Stop
```

```
_1002:
```

```
  Print "Demo ist der 2.Befehl im Menu"
```

Durch die Anweisungen im Initialisierungsteil des Hauptprogramms wird ein Menu erzeugt. Dazu werden in das String-Feld 'Menu\$' (siehe dazu Abschnitt 3.2) im ersten Feldelement die Befehlsworte des ersten Abrollmenüs geladen, 'Menu' ist der Titel des Abrollmenüs und 'Ende' und 'Demo' sind die anwählbaren Befehle innerhalb des Abrollmenüs. Das zweite Feldelement wird 'leer' gesetzt, um das Ende der Menudefinition anzuzeigen. Durch die darauffolgende Anweisung 'Menu Menu\$' wird eine Menu nach den Spezifikationen des String-Feldes 'Menu\$' generiert. Das Programm wird anschliessend suspendiert und wartet auf Messages von Windows.

Das Programm wertet nur zwei Messages aus: die Codes '1001' und '1002', die das Anwählen des ersten bzw. des zweiten Menubefehls im Abrollmenu 'Demomenu' durch den Benutzer melden. Das Programm läuft dann an den Windows-Sprungmarken '_1001' und '_1002' wieder an.

Die Codes von menubezogenen Messages werden beim Generieren des Menus festgelegt. Das erste Menu innerhalb der Menuleiste bekommt Codes beginnend mit 1000 zugeordnet, das zweite beginnend mit 2000 und so fort. Die Behle innerhalb eines Abrollmenus werden mit diesem Code beginnend durchnummeriert, wobei die Codes 1000, 2000 usw. für den Menutitel stehen. Somit erhalten im obigen Beispiel der Menutitel 'Demomenu' den Code 1000, der erste Menubefehl 'Ende' den Code 1001 und 'Demo' den Code 1002. Der Menutitel selbst erzeugt aber keine von WinBasic auswertbare Message, er sorgt intern für das Abrollen des Menus.

Entsprechend seinem Namen sorgt der Menubefehl 'Ende' für ein Beenden des Programms mit STOP nach Abarbeiten der Befehle des Unterprogramms 'ProgEnde'. Das Programm wird nicht mehr suspendiert, da STOP für ein echtes Programmende sorgt. Ein weiteres Beispiel für Menus finden Sie im Abschnitt 'Beispielprogramme' und im folgenden Abschnitt.

Aufbau einer Dialog-Routine

Ein wichtiges Element in Windows-Programmen sind Dialogboxen. Sie sind die übliche Form von Dateneingaben und Kommunikation zwischen Programm und Benutzer.

So wie das Hauptprogramm für das Bearbeiten von allgemeinen Messages verantwortlich ist, gibt es in WinBasic für jede Dialogbox ein Unterprogramm, das den Dialog steuert und dialogbezogene Messages verarbeitet.

Beispiel: Eine einfache Datenerfassung

Dim Menu\$(3)

```
Subroutine DialogDemo()  
  Dialog 10, 10, 120, 40, 0, 0, "Diademo"  
  Dialog 4, 12, 30, 14,-1, 5, "Eingabe:"  
  Dialog 40, 10, 40, 14,10,18, "ABC"  
  Dialog 90, 10, 20, 12,11,13, "OK"  
  Dialog  
  _1: 'Enter  
  _11:  
  DlgItem 10, Ein$  
  Print Ein$  
  Dialog @  
  2: 'ESC  
  Dialog @  
Endsub
```

Rem *** Hauptprogramm ***

```
Menu$(1) = "Dialog"  
Menu$(2) = "Ende"  
Menu$(3) = ""  
Menu Menu$
```

```
_1000:  
  Gosub Dialogdemo()  
_2000:  
  Stop
```

Durch die Anweisung 'Menu Menu\$' wird ein Menu erzeugt, das aus zwei einfachen Menubefehlen auf der Menuleiste besteht, d. h. beide Befehle 'Dialog' und 'Ende' sind nicht die Menutitel von Abrollmenüs sondern verhalten sich wie Menubefehle innerhalb von Abrollmenüs, sie senden also die Messages 1000 bzw. 2000 bei Anwahl durch den Benutzer.

Bei Anklicken des Menubefehls 'Dialog' verzweigt das Programm in die Subroutine 'Dialogdemo'. Diese Subroutine ist für Aufbau, Abwicklung und Beenden des Dialogs verantwortlich.

Zunächst wird durch eine Reihe von Anweisungen

```
Dialog <x%>, <y%>, <dx%>, <dy%>, <id%>, <art%>, <text$>
```

die Dialogbox generiert. Die ersten vier Parameter sind Werte fuer Anfangspunkt und Ausdehnung der einzelnen Dialogbox-Elemente oder -Items. <id%> ist die individuelle Identifikation des Items und muß innerhalb einer Dialogbox eindeutig sein (darf aber in anderen Dialogboxen auch verwendet werden). <art%> legt die Art des Items fest. Im Beispiel wird mit art%=0 in der ersten Anweisung ein Rahmen definiert, mit art%=5 in der zweiten Anweisung ein linksbündiger konstanter Text in die Box gesetzt, mit art%=18 ein Editierfeld zur Dateneingabe und mit art%=13 ein 'Druckknopf' zum Beenden des Dialogs erzeugt. (Weitere Einzelheiten im Referenzteil.)

Die Reihe von Dialog-Anweisungen wird abgeschlossen durch eine einfache Anweisung

```
Dialog
```

welche das Ende der Dialogbox-Definition anzeigt und gleichzeitig den Dialog startet. Die Subroutine wird hier suspendiert und wartet nun auf Messages von Windows. Messages werden gesendet, wenn der Programm-Benutzer Items der Dialogbox mit der Maus anwählt oder die Tasten Enter und ESC der Tastatur drückt. Wird ein Item der Dialogbox angewählt, wird als Message die Identifikation <id%> aus der Dialog-Definition gesandt, die Tasten Enter und ESC senden immer die Message-Codes 1 und 2.

Wählt der Benutzer den Druckknopf 'OK' an, wird als Message die Identifikation '11' des Druckknopfs gesandt. Durch die Anweisung DLGITEM wird das Editierfeld mit der Identifikation '10' ausgelesen und anschliessend gedruckt. Die Anweisung

```
Dialog @
```

beendet den Dialog, die Subroutine wird unverzüglich verlassen! 'Dialog @' wirkt also wie eine RETURN-Anweisung.

Detailliertere Beispiele zur Dialog-Programmierung finden Sie im Teil 'Beispielprogramme'.

3.2 Variablen in WinBasic

Daten werden in Variablen gespeichert. WinBasic unterstützt die Datentypen Integer (16-Bit Ganzzahl), Real (32-Bit Fließkommazahl), Long (32-Bit Ganzzahl), Double (64-Bit Fließkommazahl), String (Zeichenkette)

Die Variablen dürfen beliebige Namen mit max. 16 Stellen haben. Der Typ der Variablen wird durch das letzte Zeichen festgelegt, nach folgender Konvention:

% Integer,	z.B. i%
& Long,	z.B. long&
# Double,	z.B. d1#
\$ String,	z.B. str\$

Reals haben kein solches Kennzeichen. Eine Variable ist vom Typ Real, wenn sie keines der obigen Sonderzeichen als letztes Zeichen hat.

Für den Darstellungsbereich und die Genauigkeit der einzelnen Datentypen gilt das gleiche wie im Paragraphen 'Konstanten' beschrieben.

Das letzte Zeichen darf kein Punkt sein, da diese bei Records als Trennzeichen verwendet werden.

Wird innerhalb eines Programms eine Variable verwendet, die nicht (in Subroutines) ausdrücklich als LOCAL oder STATIC deklariert wurde, so handelt es sich um eine 'Global'-Variable, der Regelfall. 'Global'-Variablen sind im Hauptprogramm und in allen Subroutines bekannt, egal ob sie im Hauptprogramm oder in einer Subroutine erzeugt wurden. LOCAL- und STATIC-Variablen sind nur in der Subroutine bekannt, in der sie erzeugt wurden. Sie sind von 'Global'-variablen gleichen Namens.

Bei erstmaliger Verwendung eines Variablennamens der nicht unter LOCAL oder STATIC vordefiniert wurde legt der Compiler die Variable nach obenstehender Typen-Konvention als 'Global' an, bei weiterer Nennung des gleichen Variablen-Namens wird diese Variable verwendet.

Als zusammengesetzte Datentypen werden Felder und Records unterstützt. Felder bestehen aus gleichartigen Variablen auf die über einen gemeinsamen Namen und einen Index, der die einzelnen Variablen voneinander unterscheidet, zugegriffen. Felder werden durch die Anweisung DIM erzeugt.

Der Feld-Index wird zur Laufzeit des Programms auf Überschreitung der mit DIM vereinbarten Grenzen überwacht, wenn das Programm mit eingeschalteter 'Debug'-Option übersetzt und gestartet wird. Bei Überschreitung der Obergrenze oder Unterschreitung von 1 wird dann eine Warnung ausgegeben und die entsprechende Zeile des

Programmtexts invers dargestellt.

Records bestehen aus verschiedenen Variablen auf die über den gemeinsamen Record-Namen und einen individuellen Recordvariablen-Namen zugegriffen wird. Zwischen beide Namensteile wird ein Punkt gesetzt. Records werden zweistufig erzeugt: durch die Anweisung RECTYPE wird zunächst ein Record-Typ deklariert, auf den sich nachfolgende RECORD-Deklarationen beziehen.

Die Namen sowohl für den Record-Typ als auch für den Record müssen den Konventionen für Reals gehorchen, d. h. kein angehängtes Sonderzeichen wie %, &, #.

3.3 Konstanten

Entsprechend den verschiedenen einfachen Datentypen gibt es verschiedene Konstanten.

Integer-Konstanten besteht als ganze Zahlen einfach aus einer Anzahl von Ziffern. Die Zahlen müssen zwischen -32768 und +32767 liegen, da dieser Bereich durch 16 Bit dargestellt werden kann. Werden Integer-Konstanten ausserhalb dieses Bereichs gebildet, sind die von Übersetzer erzeugten internen Darstellungen unbestimmt. Bei positiven Zahlen kann das Plus-Zeichen entfallen.

Long-Konstanten bestehen entsprechend aus Zahlen im Bereich von -2147483648 und +2147483647 und haben zur Kennzeichnung ein 'L' angehängt, also z.B. -123L.

Real-Konstanten bestehen aus Vor- und Nachkommastellen getrennt durch einen Dezimalpunkt oder in der wissenschaftlichen Exponentialnotation aus Mantisse und Exponent, getrennt durch ein 'E'. Der gültige Bereich liegt zwischen 3.4E-38 und 3.4E38 für positive Zahlen und genauso für negative Zahlen. Die erreichbare Genauigkeit liegt bei 7 Stellen.

Für Double-Konstanten ist das 'E' der Real-Konstanten ersetzt durch 'D' und der Bereich beträgt 1.7D-308 bis 1.7D308, die Genauigkeit ca. 14 Stellen.

Da Reals und Doubles eine Fließkommazahl wegen der beschränkten Stellenzahl nicht exakt darstellen, können in der Praxis Probleme z. B. beim Vergleich von Daten auftreten. Ein offensichtliches Beispiel ist:

```
If a = 1.0/3.0 Then Break
```

Da 1/3 oder 1.0/3.0 nicht mit einer endlichen Stellenzahl als Fließkommazahl darzustellen ist, ist der Ausgang des Vergleichs unbestimmt, er kann je nach dem Weg auf dem der Wert der Variablen 'a' erzeugt wurde WAHR oder FALSCH sein.

Aus diesem Grund sollten Reals und Doubles auch nicht in FOR-Schleifen als Zählvariablen eingesetzt werden (siehe FOR).

String-Konstanten bestehen aus beliebig aneinandergereihten Zeichen, eingeschlossen durch ein Paar Anführungszeichen "". Die maximale Länge beträgt 255 Zeichen.

3.4 Arithmetische Ausdrücke

Variablen und Konstanten werden durch arithmetische Operatoren zu arithmetischen Ausdrücken verbunden, deren Wert gegebenenfalls an Variable zugewiesen wird.

Arithmetische Operatoren sind +, -, *, / entsprechend den vier Grundrechenarten und deren Regeln für Vorrang (Punktrechnung vor Strichrechnung) und Klammerung. Sind in einem arithmetischen Ausdruck mehrere Datentypen vorhanden, nimmt der Übersetzer automatisch eine Typwandlung vor, sofern es sich um numerische Datentypen handelt. Strings und Records sind mit anderen Typen nicht kompatibel, d. h. eine Wandlung ist nicht möglich. Felder können in arithmetischen Ausdrücken elementweise verwendet werden.

Bei der automatischen Wandlung wird immer in den Datentyp gewandelt der die größere Darstellungsmöglichkeiten hat, d.h. in dem Ausdruck

`a% * b1`

bestehend aus dem Integer `a%` und dem Real `b1` wird `a%` in eine Real-Zahl gewandelt und dann die Multiplikation durchgeführt.

Wird der Wert eines arithmetischen Ausdrucks durch den Zuweisungsoperator = einer Variablen zugewiesen, wie in

`erg% = a% * b1`

so wird auf jeden Fall der arithmetische Ausdruck in den Typ der Ergebnis-Variablen gewandelt. Hat dieser geringere Darstellungsmöglichkeiten, können Informationsverluste (im obigen Beispiel Verlust der Nachkommastellen) und komplette Fehlinformation durch Übersteigen der Darstellungsbreite des Ergebnistyps die Folge sein (Wenn der Ausdruck `a% * B1` z.B. 40000.0 ist, ist der Wert von `erg%` unvorhersagbar).

Werden Konstanten direkt an Variablen zugewiesen, ist es günstig wenn die Typkonventionen eingehalten werden, wie in

`a = 1.0`

`a# = 1.0D0`

Würde man dagegen schreiben

`a = 1`

`a# = 1.0`

würden die Ergebnis-Variablen die gleichen Werte erhalten, es müßte jedoch eine Wandlung durchgeführt werden, die Platz für entsprechenden Programmcode und Rechenzeit verbraucht.

Records können in arithmetischen Ausdrücken nicht als Ganzes verwendet werden, nur ihrer Elemente können verwendet werden. Ausnahme hiervon ist die Zuweisung

`Rectype Atype A1, A2`

`A1 = A2`

A1 und A2 müssen vom gleichen Record-Type sein!

Wo immer in arithmetischen Ausdrücken Variablen stehen können, dürfen auch Funktionen vom gleichen Datentyp stehen, sowohl die in WinBasic definierten Standardfunktionen wie VAL oder VAL% als auch vom Benutzer erstellte SUBROUTINES die durch RETURN einen Wert zurückgeben. Auf diese Weise können auch String-Werte in arithmetische Ausdrücke eingebracht werden.

3.5 String-Ausdrücke

String-Konstanten und -Variablen können Stringausdrücke bilden. Diese sind mit anderen Datentypen nicht kompatibel, d. h. Variablen anderer Datentypen können nicht gewandelt werden und dürfen daher in String-Ausdrücken nicht verwendet werden. Zugelassen sind Record-Elemente vom Datentyp String und Elemente aus String-Feldern.

Einziger Operator in String-Ausdrücken ist die Verkettung oder Concatanation, dargestellt durch '+' wie in

```
Erg$ = "abc" + a$ + STR$(1.23)
```

Hier ist STR\$ ein Standard-Funktion vom Typ String, die natürlich in String-Ausdrücken vorkommen dürfen. Da es Funktionen wie STR\$ gibt, welche Zahlen in Zeichenketten verwandeln, können auf diesem Weg Zahlen in Stringausdrücke eingebracht werden.

3.6 Logische Ausdrücke

Elementare logische Ausdrücke enthalten neben Variablen und Konstanten die Vergleichsoperatoren

=	gleich
<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
<>	ungleich

wie z.B. in

```
If a% > 0 Then...  
While summe% = 0
```

Der Vergleichsoperator = ist nicht zu verwechseln mit dem Zuweisungsoperator =, beide werden durch das gleiche Zeichen dargestellt, haben aber verschiedene Wirkung. In obigem Beispiel wird der Variable 'summe%' kein Wert zugewiesen, sondern es wird ein Vergleich auf Gleichheit mit der Konstanten 0 durchgeführt.

Der Wert der logischen Ausdrücke ist entweder -1, wenn der Vergleich WAHR ist oder 0, wenn er FALSCH ist.

Da logische Ausdrücke einen Wert ergeben, können sie durchaus in arithmetischen Ausdrücken verwendet werden , z.B.

```
a% = b% > 1
```

a% ist -1, wenn b% > 1, sonst 0. Andererseits können beliebige arithmetische Ausdrücke als logische Ausdrücke verwendet werden, da bei der Auswertung jeder Wert ungleich 0 als WAHR interpretiert wird. Die Schleife

```
i% = 10
While i%
  ...
  i% = i% - 1
Wend
```

wird solange durchlaufen bis i% auf 0 heruntergezählt ist und durch 'WHILE' daher als FALSCH interpretiert wird.

Logische Ausdrücke liefern jedoch den Wert -1 als Ergebnis für WAHR und zwar als Integer. -1 wurde gewählt weil in der binären Darstellung dieser Zahl alle 16 Bit auf '1' gesetzt sind.

Elementare logische Ausdrücke können durch die logischen Verknüpfungs-Operatoren 'AND', 'OR' und 'NOT' zu zusammengesetzten logischen Ausdrücken verbunden werden wie in

```
If i% > 0 AND summe > 100.0 Then ...
If NOT s$ = "123" Then ...
```

Sind mehrere Verknüpfungs-Operatoren in einem Ausdruck vorhanden, so gelten Vorrangregeln wie bei arithmetischen Ausdrücken. Eine AND-Verknüpfung wird vor einer OR-Verknüpfung durchgeführt, durch Klammerung kann die Reihenfolge geändert werden.

3.7 Datei-Bearbeitung

WinBasic verfügt über die klassischen IO-Befehle von Basic, wenn auch die Behandlung von Random-Dateien durch die Verwendung von Records etwas modernisiert wurde.

Zwei Arten von Dateien werden unterstützt:

- Text-Dateien	Open "I", "O", "U", "A"
- Random-Dateien	Open "R"

Random-Dateien haben eine feste Satzlänge. Es können nur ganze Datensätze in einen Record gelesen und aus einem Record in die Datei geschrieben werden. Als Identifikation, welcher Satz zu lesen oder zu schreiben ist, dient eine Satznummer, welche mit der Position des Satzes in der Datei identisch ist. Sätze können nur geschrieben werden, wenn sie einen bereits vorhandenen Satz überschreiben oder unmittelbar an den letzten bereits vorhanden Satz anschließen. Das heißt, Random-Dateien können nur lückenlos erzeugt werden, ein Schreiben in beliebiger Folge ist nicht möglich. Random-Dateien werden mit den Anweisungen GET und PUT bearbeitet.

Text-Dateien können Daten beliebiger Struktur enthalten. Sie werden sequentiell gelesen

oder geschrieben, d.h. vom ersten Zeichen in der Datei zeichenweise fortschreitend. Text-Dateien werden mit den Befehlen INPUT, LINE INPUT, PRINT und WRITE bearbeitet. INPUT und LINE INPUT auf der einen sowie PRINT und WRITE auf der anderen Seite bearbeiten die Dateien in etwas unterschiedlicher Art, siehe Referenzteil.

Beide Dateiararten werden mit OPEN zur Bearbeitung geöffnet und gegebenenfalls kreiert. Nach der Bearbeitung werden die Dateien mit CLOSE geschlossen.

Wird beim Öffnen einer Datei kein vollqualifizierter Dateiname angegeben - d. h. Verzeichnis und evtl. Laufwerk sind nicht ausdrücklich dem Dateinamen vorangesetzt - so wird die Datei im aktuellen Verzeichnis gesucht, welches in der Statuszeile angezeigt wird.

3.8 Zeichensätze

MS-DOS und MS-Windows arbeiten mit zwei verschiedenen Zeichensätzen: MS-DOS mit dem 'IBM Extended Character Set' und MS-Windows mit dem 'ANSI Character Set'. Bei Zeichenketten, die z. B. Umlaute enthalten, machen sich die Unterschiede sofort unangenehm bemerkbar.

Um Daten, die unter MS-DOS erzeugt wurden, unter MS-Windows verarbeiten zu können und umgekehrt unter MS-Windows MS-DOS-kompatible Daten erzeugen zu können, sind in WinBasic zwei Konvertierroutinen implementiert, nämlich WINTODOS und DOSTOWIN. Beim Öffnen von Datenbanken mit DBOPEN kann eine automatische Konvertierung angefordert werden. Näheres entnehmen Sie bitte dem Referenzteil.

3.10 Beispielprogramme

Alle Programme sind auf der Diskette in entsprechenden Dateien enthalten.

Hello.bas

Dies ist das einfachste Programm in WinBasic:

```
Print "Hello Windows"
```

Message.bas

Dieses Beispiel zeigt Ihnen wie man Menus erzeugt und die Messages verarbeitet, die von MS-Windows gesendet werden:

```
1    Dim Menu$(4)
2
3    Menu$(1)="Menu1 MenuItem_1001 MenuItem_1002"
4    Menu$(2)="&Menu2"
5    Menu$(3)="Menu3 MenuItem_3001 MenuItem_3002"
6    Menu$(4)=""
7
```

```
8      Menu Menu$
9
10     _1001:
11     _1002:
12     a$=chr$(getmessage%()-1000+48
13     MessageBox a$+". Item", "Menu1", 0, antwort%
14     _2000:
15     MessageBox "direkt wirksam", "Menu2", 0, antwort%
16
17     _3001:
18     _3002:
19     a$=chr$(getmessage%()-3000+48
20     MessageBox a$+". Item", "Menu3", 0, antwort%
21     _KEY:
22     a$=inkey$()
23     Print a$,asc%(a$)
24     _MOUSE:
25     Mouse Knopf%, x%, y%
26     MoveTo x%, y%
27     Print "Knopf ";Knopf%
```

(Dem eigentlichen Programm sind hier zur besseren Erklärung Zeilennummern vorgestellt.)

In den Zeilen 1 - 8 wird ein Menu definiert und generiert. Dazu wird ein String-Feld 'Menu\$' mit soviel Elementen definiert, wie Menupunkte auf die Menuleiste kommen sollen, plus einem zusätzlichen, bei drei Menupunkten also vier Feldelemente wie hier.

Die einzelnen Feldelemente werden mit Strings gefüllt, welche die Stichworte für die einzelnen Menubefehle enthalten. Das erste Wort ist der Menutitel, er erscheint auf der Menuleiste und läßt das Menu abrollen, wenn er angewählt wird. Die weiteren Worte sind die Menubefehle innerhalb des Abrollmenus, sie senden bei der Anwahl durch den ProgrammBenutzer Messages an das Hauptprogramm. Enthält ein String nur ein Stichwort wie in Zeile 4, wird keine Abrollmenu generiert sondern ein einzelner Menubefehl auf der Menuleiste, der ebenfalls eine Message bei Anwahl sendet.

Das letzte Feldelement wird als Leerstring "" gesetzt um das Ende der Menudefinitionen zu kennzeichnen. Durch die Anweisung in Zeile acht wird dann das Menu erzeugt und dargestellt.

Das Programm wartet nun auf Messages. Es wertet durch Bereitstellung der entsprechenden Windows-Sprungmarken Messages aus die durch das Drücken der Tastatur, Drücken einer Maustaste oder Anklicken eines Menubefehls erzeugt werden.

Beim Generieren des Menus werden den einzelnen Menubefehlen Codes zugeordnet, die sich aus Ihrer Stellung innerhalb des Menus ergeben. Dabei erhalten alle Menubefehle im ersten Abrollmenu Codes ab 1000, im zweiten Abrollmenu ab 2000 usw.. Innerhalb eines Abrollmenus werden dann alle Menubefehle beginnend ab 0 für den Menutitel durchnummeriert. Schauen Sie sich das Beispiel an, um sich den Zusammenhang zwischen der Stellung im Menu und dem zugewiesenen Code klarzumachen.

Die Menubefehle senden den ihnen zugeordneten Code als Message, sobald Sie durch den Programmbenutzer angewählt werden. Die Menutitel senden keine Messages, sondern rollen das Menu ab. Wird durch ein Element des Menudefinierenden String-Feldes (hier 'Menu\$') kein Abrollmenu definiert sondern ein einzelner Menubefehl auf der Menuleiste, senden auch dieser eine Message.

Die Windows-Sprungmarken, an denen in das Programm bei Vorliegen einer entsprechenden Message wiedereingetreten wird, ergeben sich aus dem erwarteten Code, dem ein Unterstrich '_' vorangestellt wird. Dieser Unterstrich unterscheidet auch Windows-Sprungmarken von gewöhnlichen Sprungmarken, wie sie in Basic zum Anspringen durch GOTO-Befehlen gebräuchlich sind.

Auf die Windows-Sprungmarken folgt Programmcode, der eine Reaktion auf die Message bewirkt. Bei Erreichen der nächsten Windows-Sprungmarke wird das Programm suspendiert - d. h. in Wartestellung versetzt - um auf die nächste Message zu warten. Auch bei Erreichen des physischen Programmendes (hier Zeile 27) wird das Programm nicht beendet sondern suspendiert. Nur eine explizierte STOP-Anweisung oder die Anwahl des Befehls 'Schließen/Close' im Systemmenu beendet das Programm.

Die Windows-Sprungmarken 1001 und 1002 bzw. 3001 und 3002 sind zu jeweils einer Gruppe zusammengefasst. Die Messages werden durch den gleichen Programmcode bearbeitet. Hier wird das Programm an den Stellen _1002 und _3002 nicht suspendiert, da kein Programmcode vor diesen Sprungmarken steht.

Wenn man Messages in Gruppen wie hier bearbeiten will, muß man gelegentlich doch Fallunterscheidungen im Lauf der Bearbeitungen machen. Hier ist die Funktion GETMESSAGE%() nützlich, sie gibt den Code der zuletzt empfangenen Message zurück (Siehe Zeilen 12 und 19).

Die Windows-Sprungmarken _KEY und _MOUSE sind Eintrittsstellen bei Messages, die durch Tastatur-Anschlag bzw. Maustasten verursacht werden. Sie sind neben den Menu-Messages die einzigen, die empfangen werden können.

Bei der Message KEY (Code -200 in GETMESSAGE%()) werden nur druckbare Zeichen und die Funktionstasten F1 bis F9 und Shift-F1 bis Shift-F9 berücksichtigt. Die gedrückte Taste kann durch die Funktion INKEY\$() ermittelt werden (siehe Referenzteil). Die Message MOUSE (Code -300) wird gesendet, wenn der Programm-Benutzer eine Maustaste drückt. Mit der Anweisung MOUSE können die Mausdaten wie gedrückter Knopf und die aktuellen Mauskoordinaten ermittelt werden.

Dialog.bas

Dieses Programm zeigt Ihnen, wie man Dialogboxen programmiert.

```
1    Dim Menu$(3)
2
3    Subroutine SetFont()
4        Dialog 60, 20,160,120, 0, 0, "Set Font"
5        Dialog 10, 10, 80,100, -1, 10, "Font"
```

Dokumentation WinBasic25

```

6      Dialog 20, 20, 60, 12, 11, 12, "Times"
7      Dialog 20, 32, 60, 12, 12, 12, "Helvetica"
8      Dialog 20, 44, 60, 12, 13, 12, "Swiss"
9      Dialog 20, 56, 60, 12, 14, 12, "Script"
10     Dialog 20, 68, 60, 12, 15, 12, "Roman"
11     Dialog 20, 80, 60, 12, 16, 12, "System"
12     Dialog 20, 92, 60, 12, 17, 12, "Courier"
13     Dialog 100, 20, 30, 10, -1, 5, "Höhe:"
14     Dialog 140, 18, 12, 10, 21, 18, ""
15     Dialog 100, 32, 30, 10, -1, 5, "Breite:"
16     Dialog 140, 30, 12, 10, 22, 18, ""
17     Dialog 100, 46, 50, 10, 31, 1, "Fett"
18     Dialog 100, 58, 50, 10, 32, 1, "Kursiv"
19     Dialog 100, 70, 50, 10, 33, 1, "Unterstr."
20     Dialog 100, 90, 24, 16, 3, 13, "OK"
21     Dialog 130, 90, 24, 16, 4, 11, "ESC"
22     Dialog
23     _INIT:
24     _Dlgitem 16, 11, 17, 6, chr$(1)
25     fett%=0
26     unter%=0
27     kursiv%=0
28     _1: 'Enter
29     _3: 'OK-Button
30     _Dlgitem 21, itemtext$ : h% = val%(itemtext$)
31     _Dlgitem 22, itemtext$ : w% = val%(itemtext$)
32     Font h%, w%, fett%, kursiv%+unter%*2, FF%
33     Print "Ihr Font"
34     Dialog @
35     _2: 'Escape
36     _4:
37     _Dialog @
38     _11:
39     _12:
40     _13:
41     _14:
42     _15:
43     _16:
44     _17:
45     item% = getmessage%()
47     _Dlgitem item%, 11, 17, 6, chr$(1)
48     FF%=item%-10
49     _31:
50     fett% = 1-fett%
51     _Dlgitem 31, 0, 0, fett%*4+2, chr$(1)
52     _32:
53     kursiv% = 1-kursiv%
54     _Dlgitem 32, 0, 0, kursiv%*4+2, chr$(1)
55     _33:
56     unter% = 1-unter%

```

```
57     Dlgitem 33, 0, 0, unter%*4+2, chr$(1)
58     Endsub
59
60     REM *** Hauptprogramm ***
61
62     Menu$(1)="Ende"
63     Menu$(2)="Font"
64     Menu$(3)=""
65
66     Menu Menu$
67
68     _1000:
69     Stop
70     _2000:
71     Gosub SetFont()
```

(Dem eigentlichen Programm sind hier zur besseren Erklärung Zeilennummern vorgestellt.)

Dialoge werden durch eine Subroutine komplett abgewickelt. Daher besteht das vorliegende Programm aus einem kurzen Hauptprogramm ab Zeile 60, das ein Menu generiert und dessen Messages auswertet und einer größeren Subroutine für die Dialogbox. Die Subroutine hat drei Aufgaben:

- Erzeugen der Dialogbox
- Verarbeiten der dialogbezogenen Messages
- Beenden des Dialogs

Hier ist eine Dialogbox beschrieben, die einen beliebigen Font aus den verfügbaren Font-Familien generiert. Dazu muß die Font-Familie ausgewählt werden, Höhe und Breite des Fonts sowie spezielle Auszeichnungen (fett, kursiv, unterstrichen) müssen festgelegt werden.

Zunächst wird die Dialogbox mit Anweisungen der Form
Dialog <x%>, <y%>, <dx%>, <dy%>, <id%>, <art%>, <text\$>
generiert. <x%> und <y%> sind die Koordinaten der linken oberen Ecke des jeweils zu erzeugenden Elements oder Items der Dialogbox, <dx%> und <dy%> die Ausdehnung des Elements in x- und y-Richtung. <id%> ist die Identifikation des Elements innerhalb der Dialogbox, diese muss innerhalb einer Dialogbox eindeutig sein, da sie als Code für Messages verwendet wird. <art%> legt die Art des zu erzeugenden Dialog-Items fest: konstanter Text, Eingabefeld, Druckknopf usw.. Mit <text\$> wird das Item beschriftet. Eine weitergehende Erläuterung zu den Parametern der Anweisung DIALOG sollten Sie sich unbedingt im Referenzteil anschauen.

Der erste Aufruf von DIALOG generiert mit mit art%=0 einen Dialog-Rahmen. Mit text%="Set Font" wird eine Überschrift in diesen Rahmen gesetzt. <id%> ist ohne Bedeutung, da der Rahmen keine Messages sendet und sollte '0' gesetzt werden. <x%> und <y%> legen hier die linke obere Ecke des Rahmens innerhalb des Programm-Fenster fest.

Die weiteren DIALOG-Anweisungen definieren die einzelnen Elemente der Dialogbox. `<x%>` und `<y%>` legen hier die linke obere Ecke des Elements innerhalb des Dialog-Rahmens fest. `<id%>`.

Zunächst wird mit `art%=10` eine sogenannte Groupbox erzeugt, das ist ein Linienrahmen mit `<text$>` als Titel. Die Identifikation `<id%>` ist ohne Belang, da der Linienrahmen vom Programm-Benutzer nicht angewählt werden kann und daher auch keine Message sendet, daher ist sie `'-1'` gesetzt.

Der Rahmen umfasst die nachfolgenden sieben Leuchtknöpfe, die durch `art%=12` erzeugt werden, und unterstreicht so ihre Zusammengehörigkeit in einer Gruppe. Die Leuchtknöpfe haben aufeinanderfolgende Identifikationen 11 bis 17, was bei der Steuerung des Dialoges wichtig ist, denn nur so kann mit einer DLGITEM-Anweisung ein Knopf aus dieser Gruppe zum Leuchten gebracht werden, während alle anderen ausgeschaltet sind.

Dann folgen zwei Paare von konstantem linksbündigen Text (`art%=5`) und einem linksbündigem Editierfeld (`art%=18`). Die Texte haben als Identifikation `'-1'`, da sie nicht angewählt werden können und daher keine Message senden.

Um die Auszeichnungen 'fett', 'kursiv' und 'unterstrichen' festzulegen werden mit `art%=1` drei Checkboxes generiert. Diese Checkboxes tragen entweder Checkmarken oder nicht und werden für ja/nein-Entscheidungen verwendet.

Es folgen zwei Druckknöpfe. Mit `art%=13` wird der Standard-Druckknopf "OK" erzeugt, mit `art%=11` der Druckknopf "ESC". Der Standard-Druckknopf unterscheidet sich vom einfachen Druckknopf lediglich durch die stärkere Unterrandung. Der Dialog sollte so programmiert werden, dass seine Auswahl dieselbe Reaktion wie das Drücken der Enter-Taste bewirkt.

Nun, da alle Elemente definiert sind, wird mit DIALOG ohne Parameter die Reihe der Definitionen abgeschlossen und die Dialogbox auf den Bildschirm gebracht. Tastatur-Eingaben durch den Benutzer werden anfangs in das erste Editierfeld oder den Standard-Druckknopf gelenkt, falls dieser früher in der Definitions-Reihenfolge stand. Dieses Dialog-Item hat damit den sogenannten Input-Focus. Dieser kann mit der Maus oder den TAB-Tasten auf andere Items gelegt werden. Dann werden Tastatur-Eingaben in das neue Item gelenkt. Das Item, welches den Input-Focus hat, ist entweder mit einem punktierten Linien-Rahmen oder einem Text-Cursor (bei Editierfeldern) gekennzeichnet.

Wählt der Benutzer eins der Dialog-Items durch Anklicken mit der Maus oder durch Drücken der Leertaste an, so sendet das Item seine bei der Erzeugung vergebene Identifikation als Message. Um auf die Message reagieren zu können enthält die Subroutine entsprechende Windows-Sprungmarken als Programm-Eintrittspunkte. Sie bestehen aus dem Identifikations-Code und einem vorangestellten Unterstrich `'_'`, analog zu den menuorientierten Sprungmarken im Hauptprogramm. Es gibt noch eine zusätzliche Message: INIT. Diese Message wird gesandt kurz bevor die Dialogbox auf den Bildschirm gebracht wird. Hier kann eine Initialisierung von Items vorgenommen werden. Im vorliegenden Beispiel wird durch die Zeile

Der Leuchtknopf 16 in der Gruppe der Leuchtknöpfe 11 bis 17 zum Leuchten gebracht und es werden die Merker für die Checkboxes initialisiert. Durch `text$=chr$(1)` bleibt die vorhandene Beschriftung bestehen.

Die Messages 11 bis 17 der Leuchtknöpfe sind in einer Gruppe zusammengefasst. Um zu erkennen, welcher Leuchtknopf angewählt wurde wird die Funktion `GETMESSAGE%()` verwendet. Dieser Leuchtknopf wird dann zum Leuchten gebracht. Die Identifikationen der Leuchtknöpfe wurde so gewählt, daß der Code für die zugeordnete Font-Familie - wie er für die Anweisung `FONT` benötigt wird - durch Subtraktion von 10 ergibt.

Wird eine der drei Checkboxes 31 bis 33 angewählt, so wird der zugehörige Merker zwischen '0' und '1' hin- und hergeschaltet und in Abhängigkeit von seinem Wert die Checkmarke gesetzt oder gelöscht.

Von den Editierfeldern werden keine Messages ausgewertet. Dies sollte in der Regel unterlassen werden, da sie von Windows abgehandelt werden.

Erst wenn die Messages 1 (Enter) oder 3 "OK" kommen, werden die Editierfelder ausgelesen und ihre Werte ermittelt. Zusammen mit den anderen Parametern wird durch die `FONT`-Anweisung ein Font generiert und ein Text ausgegeben. Die Dialogbox wird durch

Dialog @

beendet und die Subroutine unverzüglich verlassen wie bei einer `RETURN`-Anweisung.

Bei Eingang der Messages 2 (ESC) oder 4 "ESC" wird die Dialogbox ohne weitere Aktion beendet.