MS-DOS - Software für TurboPascal 5.0 / 5.5

# ***ASV - Application Services v1.0***

# I Allgemeines

**1. Einführung**

*ASV* ist eine Unit für *TurboPascal 5.0* und *5.5*, mit der Pro-grammierer leicht eine komplexe Menüsteuerung in eigene Programme einbinden können. Neben Pulldown-Menüs können Popup-Menüs, Alert- und Dialogboxen verwaltet werden, deren Bedienung auch komplett mit Hotkeys oder einer Microsoft-kompatiblen Maus möglich ist.

Alle späteren Versionen der *Application Services* werden zu der vorliegenden Version kompatibel sein, d.h. die alten Quelltexte brauchen nicht umprogrammiert zu werden, man muß lediglich die neue Unit beim Compilieren verwenden.

Eingetragene Benutzer können neue Versionen der *Application Services* im Rahmen des Update-Services günstig erwerben. Sie werden entsprechend benachrichtigt. Eingetragener Benutzer werden Sie, indem Sie den beiliegenden Lizenzvertrag ausgefüllt und unterschrieben an **Soft*design*** zurücksenden.



Das Handbuch wurde mit Microsoft *WORKS* erstellt und auf einem Brother *M-1724L*-Drucker ausgedruckt. Für alle Grafiken wurde *PaintShop plus* auf einem Atari ST verwendet.



*1.1 Lizenzvertrag*

a) Produkte von **Soft*design* \\\ Thomas Much Computer Software**

(nachfolgend "**Soft*design***") sind urheberrechtlich geschützt.

b) Mit dem Erwerb dieses Produktes von **Soft*design*** wird dem Käufer ein einfaches Recht zur Benutzung des urheberrechtsgeschützten Produktes eingeräumt, d.h. die Benutzung des Programms darf zur gleichen Zeit nur an einem Ort auf einem Computer möglich sein.

c) Zur Programmsicherung ist es dem Käufer erlaubt, eine Kopie des Programms anzufertigen. An der Kopie muß der Copyright-Vermerk fest angebracht werden. Das Kopieren auf Festplatte ist ebenfalls erlaubt, sofern die unter b) genannten Bestimmungen eingehalten werden.

d) **Soft*design*** übernimmt keine Gewähr für das unterbrechungs- oder fehlerfreie Funktionieren der Software oder die Genauigkeit der Daten. Es wird auch keine Gewährleistung dafür übernommen, daß das Programm den Anforderungen des Erwerbers entspricht. Die Haftung für unmittelbare und mittelbare Schäden, Folge- und Drittschäden ist - soweit gesetzlich zulässig - ausgeschlossen.
Die Haftung bei grober Fahrlässigkeit und Vorsatz ist in jedem Fall beschränkt auf den Kaufpreis.

e) Der Programmcode oder Teile davon dürfen in keiner Weise verändert oder an Dritte weitergegeben werden. Davon ausgenommen ist die Weitergabe des Programmcodes der Unit *ASV*, sofern diese in ein compiliertes Programm eingebunden ist und die Daten und Routinen der Unit nicht mehr von außen zugänglich sind.

f) Programme, die mit den *Application Services* erstellt wurden,

dürfen ohne Lizenzgebühren weitergegeben und vertrieben werden, sofern die unter e) genannten Bestimmungen eingehalten werden. Die Verwendung der *Application Services* muß nicht angegeben werden.

g) Schulversionen dürfen nicht zur Entwicklung kommerzieller Programme eingesetzt werden.

## *1.2 Installation*

Auf der Diskette befinden sich folgende Dateien:

*ASV.TPU*

Die eigentliche Unit, die in eigene Pascal-Programme eingebunden werden muß.

*WORKINIT.INC*

Diese Datei kann mit ^K^R im TP-Editor hinzugeladen werden. Sie vereinfacht das Initialisieren von Menüstrukturen.

*HILFE.INC*

Diese Datei stellt das Grundgerüst für die Ausgabe von Hilfsmeldungen dar (siehe auch *WORKINIT.INC*).

*MENUDEMO.PAS*

Ein Beispielprogramm für die Verwendung der *ASV*-Routinen. Dieses Programm ist auch in *Anhang C* abgedruckt und näher erläutert.

*MENUDEMO.EXE*

Das läuffahige Programm zu *MENUDEMO.PAS*.

*LIESMICH* (evtl.)

In dieser Datei sind Änderungen und Erweiterungen gegen-
über dem Handbuch vermerkt.

Bitte fertigen Sie sich zuerst eine Sicherheitskopie der ASV-Originaldiskette an ! *Arbeiten Sie nur noch mit dieser Kopie.* Sie können die Dateien allerdings auch auf Festplatte (sofern Sie eine besitzen) kopieren.

Die Datei *ASV.TPU* sollte in das TP-Unit-Verzeichnis kopiert werden, die Dateien *WORKINIT.INC*, *HILFE.INC* und *MENUDEMO.PAS* kopieren Sie am besten in Ihr Standard-TP-Verzeichnis.
Die *Application Services* können Sie nutzen, wenn Sie diese folgendermaßen in Ihr Pascal-Programm einbinden:

```
program XYZ;
 uses asv;
 ...
begin
 ...
end.
```

*1.3 Hotline*

Eine Hotline steht Dienstags und Freitags in der Zeit von 18 Uhr bis 20 Uhr zur Verfügung. Die Hotline erreichen Sie unter **(05121) 26 20 07**.

Bei längeren Fragen bzw. wenn ausführlichere oder schriftliche Antworten gewünscht sind, wenden Sie sich bitte an:

**Soft*design*** Computer Software * Koppelweg 16a * D-3201 Diekholzen

## 2. Reservierte Variablen

*ASV* stellt einige Konstanten zur Verfügung, die zur Pro-grammierung verwendet werden sollten, damit alle Programme zu späteren Versionen der *Application Services* kompatibel sind. Die hier vorgestellten Variablen haben allgemeine Bedeutung, alle spezifischen Konstanten sind bei den entsprechenden Routinen auf-geführt.

***_asv*** = 1.0 *ASV*-Versionsnummer

***_magic*** = 18011972 Magic-Konstante der *Application Services*

***_wmax*** = 99 Länge des work[]-Feldes-1

***_imax*** = 9 Länge des intout[]-Feldes-1

***_bmax*** = 14 Länge des byteout[]-Feldes-1

***work*** *: array [0.._wmax] of string*

Dieses String-Array wird für die Prozedur *menu_init* und die Funktionen *menu_pop()* und *dialog* benötigt.

***intout*** *: array [0.._imax] of integer*

Dieses Integer - Array wird von der Prozedur *menu_inquire* als Ausgabefeld benutzt.

***byteout*** *: array [0.._bmax] of byte*

Auch dieses Byte-Array dient als Ausgabefeld für *menu_inquire*.

Alle drei Arrays können auch für eigene Zwecke verwendet werden. *byteout[0..4]* und *intout[0..4]* werden allerdings von vielen Routinen verändert. Der Rest wird - außer von *menu_inquire* - nicht verändert.

***void.x*** : Diese Variable nimmt Werte von Funktionen auf, deren Rückgabe nicht benötigt wird. Man darf sich deshalb auch nicht darauf verlassen, daß die Variable zu einem Zeitpunkt einen bestimmten Wert besitzt.
Mögliche Endungen: ***b***=*Byte*, ***si***=*ShortInt*, ***i***=*Integer*, ***w***=*Word*, ***l***=*LongInt*, ***bo***=*Boolean*, ***c***=*Char*, ***s***=*String*, ***r***=*Real*, ***sr***=*Single(ShortReal)*, ***d***=*Double*, ***e***=*Extended*, ***co***=*Comp*

*Beispiel:* *void.i:=form_alert('TEST','Das ist ein Test.',' OK ',0)*

# II Menüverwaltung

## 1. Pulldown-Menüs

### 1.1 Bedienung

Wenn die nicht speziell markierte Menüleiste am oberen Bild-schirmrand zu sehen ist und sich keine Dialog- oder Alertboxen o.ä. auf dem Bildschirm befinden, kann mit *<Alternate>* das Menü aktiviert werden (es ist natürlich möglich, daß das jeweilige Pro-gramm das Aufrufen des Menüs verhindert, z.B. wenn es sich in einer eigenen Unterroutine befindet). Der Titel des aktuellen Pulldown-Menüs wird nun schwarz hinterlegt. Das aktuelle Menü ist das zuletzt benutzte bzw. - am Programmanfang - das Menü ganz links. Ein anderes Menü kann mit *<Cursor links>* bzw. *<Cursor rechts>* angewählt

werden. Mit *<Cursor hoch>*, *<Cursor runter>*, *<Return>* oder *<Enter>* kann das Pulldown-Menü nun aufgeklappt werden (außer wenn es sich um ein leeres Menü handelt). Man kann aber auch die hell dargestellten *Hotkeys* verwenden, um ein Menü anzuwählen und aufzuklappen. Mit *<Esc>* oder *<Alternate>* ist es möglich, das Menü wieder zu deaktivieren. *<F10>* bewirkt dasselbe wie *<Alternate>*, außer daß das aktive Menü automatisch aufgeklappt wird.

Auch im geöffneten Pulldown-Menü kann mit *<Cursor links>* und *<Cursor rechts>* das Menü gewechselt werden. Mit *<Cursor hoch>* bzw. *<Cursor runter>* kann ein anderer Menüeintrag angewählt werden, der schwarz hinterlegt wird. Der aktuelle Eintrag kann mit *<Return>* bzw. *<Enter>* angewählt werden, mit den markierten *Hotkeys* kann man jeden verfügbaren Eintrag selektieren. Mit *<Esc>* kann das Menü jederzeit deaktiviert werden, *<Alternate>* führt zur aktivierten Menüleiste zurück.

Mit einer **Maus** können die einzelnen Menüs jederzeit durch An-klicken des entsprechenden Menütitels mit der *linken Maustaste* aktiviert und aufgeklappt werden. Die Menüeinträge werden eben-falls durch Anklicken mit der *linken Maustaste* selektiert. Durch einen Klick mit der *linken Maustaste* außerhalb des Menüs kann dieses deaktiviert werden.

Ist das jeweilige Programm nach den Richtlinien programmiert, erscheinen bei den einzelnen Menüeinträgen kurze Hilfsmeldungen am unteren Bildschirmrand.

**1.2 Befehle**

### 1.2.1 Prozedur ***menu_init***

***procedure menu_init;***

```
Seite &s
```

Bevor ein Pulldown-Menü benutzt werden kann, muß es mit *menu_init* initialisiert und angemeldet werden. Die Daten für Menütitel und Menüeinträge werden dabei in das work[]-Feld geschrieben und dürfen maximal 30 Zeichen lang sein. Der Aufbau des Feldes sieht dabei wie folgt aus:

```
work[0]:='#Menü_1';          Menütitel 1
work[1]:='erster #Eintrag';  die Einträge von Menü 1 ...
work[2]:='#letzter Eintrag';  ...
work[3]:='';                 ein Leerstring zwischen den einzelnen Menüs
work[4]:='Menü_#2';          Menütitel 2
work[5]:='#erster Eintrag';  die Einträge von Menü 2 ...
work[7]:='-';                so wird eine Trennlinie programmiert
work[8]:='letzter E#intrag';  ...
work[9]:='';                 zwei Leerstrings am Ende der gesamten Struktur
work[10]:='';
```

Es werden bis zu 20 Menüs unterstützt, die theoretisch beliebig lang sein können, allerdings ist durch die Anzahl der darstell-baren Zeilen eine Grenze gesetzt (meistens 25, siehe auch *Funktion _scry*). Die Anzahl der work[]-Einträge beträgt z.Z. 98 (*work [0.._wmax-2]*). **Hotkeys** können durch ein vorangestelltes Doppelkreuz ('*#*') definiert werden; man muß selbst darauf achten, daß die Hotkeys innerhalb eines Menüs nicht doppelt vorkommen.

Nach dem Aufruf von *menu_init* stehen alle Daten in geschützten Speicherbereichen, so daß das work[]-Feld nun anderweitig genutzt werden kann (z.B. von Dialogboxen). *menu_init* übergibt in *asverror* den Fehler *E_MUSED*, wenn noch ein altes Menü aktiv ist (dieses muß dann mit *menu_kill* abgemeldet werden).

Zur einfacheren Programmierung kann die Datei *WORKINIT.INC* im Editor mit ^K^R an der aktuellen Cursorposition dazugeladen werden. Diese Datei stellt ein leeres work[]-Feld innerhalb der Prozedur

*work_init* bereit, das dann nur noch entsprechend gefüllt werden muß; nicht benötigte Einträge können zum Schluß gelöscht werden. Schließlich muß man noch beachten, daß *work_init* **vor** *menu_init* aufgerufen wird.

## 1.2.2 Prozedur ***menu_draw***

***procedure menu_draw;***

Diese Prozedur zeichnet die Menüleiste am oberen Bildschirmrand. Der Hintergrund wird dabei gerettet und später von *menu_kill* wieder restauriert. *menu_draw* kann beliebig oft aufgerufen werden, wenn z.B. die Menüleiste durch eigene Routinen zerstört wurde. Ist die Menüstruktur beim Aufruf der Prozedur noch nicht initiali-siert, wird in *asverror* der Fehler *E_NINIT* übergeben.

## 1.2.3 Funktion ***menu_watch***

***function menu_watch : integer;***

Nach dem Zeichnen der Menüleiste wird mit dieser Funktion die gesamte Menüverwaltung durchgeführt. Der weitere Programmverlauf muß von der Rückgabe dieser Funktion abhängig gemacht werden. Stellt die Routine fest, daß *<Alternate>* oder *<F10>* gedrückt wird, aktiviert sie das Menü und läßt den Benutzer einen Eintrag aus-wählen (siehe 1.1). Wurde ein Eintrag angewählt, liefert die Funktion einen Wert größer gleich *M_OK* zurück. Dieser Wert entspricht dem jeweiligen work[]-Eintrag und muß nun verarbeitet werden. Beim Abbruch des Menüs erhält man *M_CANCEL*.

Wurde das Menü allerdings nicht aktiviert, ist der Rückgabewert *M_NOP*, und andere Routinen des Programms können ausgeführt werden.

Man erfährt von *menu_watch* auch, ob und welche Taste gedrückt wurde. Ist *byteout[0]* gleich *KEY_NONE*, wurde keine Taste betätigt. *KEY_NORM* bedeutet, daß eine "normale" Taste gedrückt wurde, der Tastencode steht in *byteout[1]*. *KEY_SPEC* schließlich kennzeichnet eine Sondertaste (F9, PgUp etc.). Auch hier steht der Tastencode in *byteout[1]*. Eine Tabelle der Sondertastencodes befindet sich in *Anhang A*.

*Beispiel:* *ivar:=menu_watch;*
*if byteout[0]=KEY_SPEC then writeln('Es wurde eine Sondertaste gedrückt.');*

*Anhang C* liefert ein ausführliches Programmbeispiel. Eventuelle Fehlermeldungen sind in *V,7* beschrieben.

### 1.2.4 Prozedur ***menu_kill***

***procedure menu_kill;***

Mit dieser Routine kann eine Menüstruktur wieder abgemeldet werden (z.B. am Programmende oder wenn ein neues Menü angemeldet werden soll). Der ursprüngliche Hintergrund wird dabei wieder her-gestellt. In *asverror* wird *E_NDRAW* übergeben, wenn gar kein Menü aktiv ist, das gelöscht werden könnte.

## 1.2.5 Prozedur *menu_add()*

***procedure menu_add(wrk : string);***

*menu_add* hängt im letzten Menü einen Eintrag an die Menüstruktur an. Das Format des Strings *wrk* entspricht einem normalen Menü-eintrag im work[]-Feld (also mit Hotkey). Diese Routine sollte dazu benutzt werden, Daten darzustellen, die erst zur Laufzeit des Programms bekannt werden (z.B. geladene Dokumente in einer Text-verarbeitung). Ist das work[]-Feld voll oder der übergebene String leer, erhält man die Fehlermeldung *E_MSTRUC*.

*Beispiel:* *menu_add('#5 TEXT_5.DOC');*

## 1.2.6 Prozedur *menu_delete()*

***procedure menu_delete(num : byte);***

Diese Prozedur löscht einen Eintrag aus dem letzten Pulldown-Menü. *num* bezeichnet dabei die Nummer des zu Löschenden work[]-Eintrags. *Achtung: Die Nummern aller folgenden Einträge erniedri-gen sich dabei automatisch um 1.* Diese Routine dient dazu, nicht mehr verwendete Daten auch optisch verschwinden zu lassen (z.B. wenn in einem Editor ein Text gelöscht wurde). Gehört der zu Löschende Eintrag nicht zum letzten Menü oder soll ein leerer Eintrag gelöscht werden, erscheint die Fehlermeldung *E_MSTRUC*.

## 1.2.7 Prozedur ***menu_rename()***

***procedure menu_rename(num : byte; wrk : string);***

Mit *menu_rename* können Menüeinträge eines bereits initiali-sierten und gezeichneten Menüs umbenannt werden. Menüeinträge zeigen dadurch immer den aktuellen Zustand einer Funktion an, es wird also *kontext-sensitive* Programmierung erreicht. *num* gibt die work[]-Nummer des Eintrags an, *wrk* enthält den neuen Text (incl. Hotkey; dieser muß nicht mit dem alten identisch sein). Ist der übergebene String leer oder soll ein leerer Eintrag bzw. ein Menütitel umbenannt werden, erscheint die Fehlermeldung *E_MSTRUC*.

## 1.2.8 Prozedur ***menu_attr()***

***procedure menu_attr(num,attrs : byte);***

*menu_attr* versieht einzelne Menüeinträge mit bestimmten Attri-buten. *num* gibt dabei die Nummer des entsprechenden Eintrags an. Die folgende Tabelle zeigt alle möglichen Werte für *attrs*:

*A_CHECKED*: Kennzeichnet einen Eintrag mit einem Punkt.
*A_NCHECKED*: Löscht den Punkt vor einem Eintrag
*A_DISABLED*: Macht einen Eintrag nicht anwählbar.
*A_ENABLED*: Ermöglicht das Anwählen eines Eintrags wieder.

### 1.2.9 Funktion *menu_stat()*

***function menu_stat(num : byte) : byte;***

*menu_stat* liefert das Attribut des mit *num* angegebenen Eintrags zurück. Mögliche Werte sind *0* (kein Attribut gesetzt), *A_CHECKED* und *A_DISABLED*.

### 1.2.10 Funktion *menu_text()*

***function menu_text(num : byte) : string;***

Von dieser Funktion wird der aktuelle Text des mit *num* be-stimmten Eintrags zurückgegeben. Allerdings werden die Doppel-kreuze der Hotkeys automatisch entfernt (statt '*Eintrag #1*' erhält man also '*Eintrag 1*').

### 1.2.11 Prozedur *menu_inquire*

***procedure menu_inquire;***

*menu_inquire* liefert in *byteout[]* und *intout[]* einige Werte zurück, die für eine weiterreichende Programmierung interessant sein könnten:

```
Seite &s
```

byteout[0] : Farbwert für normale Schrift

byteout[1] : Farbwert für invertierte Schrift

byteout[2] : Farbwert für helle Schrift

byteout[3] : Farbwert für helle invertierte Schrift

byteout[4] : Farbwert für nicht anwählbare Einträge

byteout[5] : Farbwert des Schattens

Zu den Farbangaben siehe auch *Prozedur menu_setc()*.

byteout[6] : *reserviert, immer 0*

byteout[7] : " "

byteout[8] : " "

byteout[9] : " "

byteout[10] : Anzahl der Maustasten (0,2 oder 3)

byteout[11] : Computertyp (PC,XT bzw. AT)

*Beispiel:* *menu_inquire;*

*if byteout[11]=AT then writeln('Der Computer ist ein AT.');*

byteout[12] : *reserviert*

byteout[13] : "

byteout[14] : "

intout[0] : aktive Länderkennung (L_USA,L_FRG oder L_GB)

intout[1] : Bitmaske aller verfügbaren Länderkennungen

intout[2] : niedrigste verfügbare Fehlernummer

intout[3] : Speicherausbau in KByte (ohne EMS)

intout[4] : *reserviert*

intout[5] : "

intout[6] : "

intout[7] : Nummer des letzten Menüeintrags

*Beispiel:* *menu_inquire;*

*menu_delete(intout[7]); { löscht den letzten Menüeintrag }*

intout[8] : *reserviert, immer 0*

intout[9] : " "

## 2. Popup-Menüs

### 2.1 Bedienung

In einem Popup-Menü kann der aktuelle Eintrag, der schwarz hinterlegt ist, mit den Tasten *<Cursor hoch>* und *<Cursor runter>* gewechselt werden. *<Return>* oder *<Enter>* selektieren den aktuellen Menüeintrag, mit den *Hotkeys* kann jeder verfügbare Eintrag angewählt werden. *<Esc>* verläßt das Menü. Mit einer **Maus** kann ein Eintrag durch Anklicken mit der *linken Maustaste* angewählt werden, ein Klick außerhalb des Menüs bricht dieses ab.

Auch bei einem Popup-Menü sollte bei jedem Menüeintrag eine Hilfsmeldung am unteren Bildschirmrand erscheinen.

### 2.2 Befehle

```
Seite &s
```

## 2.2.1 Funktion ***menu_pop()***

***function menu_pop(wo : byte) : integer;***

Auch Popup-Menüs werden über das work[]-Feld programmiert. Damit die Daten von mehreren Menüs gleichzeitig in work[] gespeichert sein können, kann der Anfang eines jeden Popup-Menüs beim Funk-tionsaufruf mit der Variablen *wo* übergeben werden. Außerdem kann man so auch Kollisionen mit den Daten von Dialogboxen vermeiden. Prinzipiell sind Popup-Menüs wie folgt aufgebaut:

work[20]:=chr(x_pos)+chr(y_pos)+chr(default)+chr(flag)

*x_pos*[1.._scrx] und *y_pos*[1.._scry] geben die Position des Popup-Menüs an. #255 bedeutet dabei, daß das Menü auf der jeweiligen Achse zentriert wird. *default* bezeichnet den Eintrag, der beim Aufruf automatisch aktiv ist (in diesem Fall #0 für work[22] etc.). *flag* muß z.Z. immer auf #0 gesetzt werden.

work[21]:='TITEL'; Der *Titel* des Popup-Menüs.

work[22]:=#0'Eintrag #1';

Das erste Zeichen jedes Eintrags ist das *Attribut*, hier ist keins gesetzt (*#0*). *Hotkeys* werden wie üblich durch ein vorangestelltes Doppelkreuz gekennzeichnet.

work[23]:='-'; Das ist eine *Trennlinie*.

work[24]:=chr(A_CHECKED)+'#Eintrag 2';

Hier wird das *Attribut A_CHECKED* verwendet, d.h. es befindet sich ein Punkt vor dem Eintrag, der einen aktiven Status kennzeichnen soll. Als weiteres Attribut ist *A_DISABLED* möglich, der entsprechende Eintrag ist dann nicht anwählbar. Es ist auch die Kombination beider Attribute zulässig (*chr(A_CHECKED+A_DISABLED)*).

work[25]:=''; Zum Schluß zwei Leerstrings.

work[26]:='';

Der Aufruf würde bei diesem Beispiel also lauten: *ivar:=menu_pop(20);*.

Wurde ein Eintrag angewählt, erhält man einen Wert größer gleich *M_OK* zurück. 0 würde in diesem Fall also work[22] entsprechen, 2 wäre work[24]. Beim Abbruch des Menüs erhält man *M_CANCEL*.

Mögliche Fehlermeldungen sind in *V,7* beschrieben.

Die Daten im work[]-Feld werden nicht verändert, so daß man nach dem Funktionsaufruf die Werte für einen erneuten Aufruf ändern kann. Man kann z.B. den angewählten Button als neuen Default-Button einsetzen:

*Beispiel:* *ret:=menu_pop(anfang);*
*if ret>=M_OK then work[anfang,3]:=chr(ret);*

## 3. Attributfunktionen

### 3.1 Prozedur *menu_setc()*

***procedure menu_setc(was,saw,has,haw,dis,shdw : byte);***

Mit dieser Prozedur können die Farben für Pulldown-Menüs etc. eingestellt werden. *was* bezeichnet dabei normalen Text auf schwarzem Hintergrund, *saw* schwarzen Text auf invertiertem Hintergrund, *has* hellen Text auf normalem Hintergrund und *haw* hellen Text auf invertiertem Hintergrund. Mit *dis* wird der Farbwert für nicht anwählbare Menüeinträge bestimmt, *shdw* gibt die Farbe des Schattens an.

Alle Farbbytes bestehen aus Vorder- und Hintergrundbits. Die Farbbytes können mit der Funktion *attr()* recht einfach berechnet werden.

## 3.2 Prozedur ***menu_resetc***

***procedure menu_resetc;***

*menu_resetc* stellt Standard-Farbwerte für Pulldown-Menüs etc. ein. Im monochromen Modus sind dies *#7*, *#112*, *#15*, *#127*, *#127* und *#0*, im Farbmodus *attr(lightgray,black)*, *attr(black,lightgray)*, *attr(green,black)*, *attr(red,lightgray)*, *attr(brown,lightgray)* und *attr(black,black)*.

## 3.3 Prozedur ***menu_setp()***

***procedure menu_setp(procv : types.procvar);***

Mit dieser Routine kann eine Prozedur definiert werden, die von den *Application Services* beim Anwählen von Menüeinträgen etc. aufgerufen wird. Diese Prozedur ist nun selbst dafür verantwortlich, eine Hilfsmeldung anzuzeigen. Sie muß außerdem vom folgenden Typ sein:

```
Seite &s
```

*{$F+} procedure xyz(wnum : byte); {$F-}*

*begin*

*end;*

Der Name der Prozedur ist beliebig, wichtig dabei ist nur, daß sie als *Far-Call* compiliert wird (s. *Compilerdirektiven*). Der Aufruf würde nun mit *menu_setp(xyz);* erfolgen.

*wnum* kann vier verschiedene Zustände annehmen: *0*<=*wnum*<*P_OFFS*, *wnum*>=*P_OFFS*, *wnum=M_HELP* und *wnum=M_EXIT*. Im ersten Fall wird in *wnum* die Nummer des Pulldown-Menü-Eintrags übergeben, für den eine Hilfsmeldung am unteren Bildschirmrand erscheinen soll. Im zweiten Fall soll eine Hilfsmeldung für einen Eintrag des gerade aktiven Popup-Menüs ausgegeben werden. *(wnum-P_OFFS)* entspricht dabei dem jeweiligen Eintrag (0=erster Eintrag etc.). ***M_HELP*** signalisiert, daß in einem Menü etc. die Taste *<F1>* gedrückt wurde und aus-führliche Informationen gewünscht sind. In *byteout[2]* steht nun, aus welcher Struktur die Hilfsfunktion aufgerufen wurde. *H_PULL* bezeichnet das Pulldown-Menü, *H_POP* ein Popup-Menü; in *byteout[3]* steht jeweils die Nummer des aktuellen Eintrags. *H_DIAL* steht für Alert- und Dialogboxen, in *byteout[3]* erhält man eine Nummer, die bei den Funktionsaufrufen *form_alert()* bzw. *dialog* gesetzt werden kann. Die erweiterten Hilfsfunktionen sind selbst dafür verant-wortlich, daß ihre Ausgaben wieder korrekt vom Bildschirm ver-schwinden.

***M_EXIT*** bedeutet, daß ein Pulldown-Menü etc. verlassen wurde und daß die unterste Bildschirmzeile nun zu löschen ist (es können allerdings auch allgemeine Hilfen angegeben werden).

Ein Grundgerüst für eine Hilfsfunktion steht in der Datei *HILFE.INC* zur Verfügung, die mit ^K^R im Editor nachgeladen werden kann. Ein ausführliches Programmbeispiel zur Programmierung einer Hilfsfunktion findet sich in *Anhang C*.

## 3.4 Prozedur ***menu_resetp***

***procedure menu_resetp;***

*menu_resetp* schaltet die Hilfsfunktion aus, eine evtl. zuvor mit *menu_setp()* definierte Prozedur wird nun nicht mehr aufgerufen.

# III Dialogverwaltung

**1. Bedienung**

Alert- und Dialogboxen werden auf gleiche Art und Weise bedient. Am unteren Rand der Box

befinden sich mehr oder weniger *Buttons* (mind. aber einer), von denen der aktive von hellen Klammern um-geben ist. Mit *<Tab>* kann der aktuelle Button von links nach rechts gewechselt werden, mit *<Shift>+<Tab>* von rechts nach links. *<Return>* bzw. *<Enter>* selektiert den aktiven Button, *<Esc>* verläßt die Box. Außerdem kann jeder Button mit dem hell markierten *Hotkey* angewählt werden. Mit einer **Maus** kann jederzeit jeder verfügbare Button durch Anklicken mit der *linken Maustaste* selektiert werden.

**2. Befehle**

## 2.1 Funktion *form_alert()*

***function form_alert(titl,msg,but : string;def : byte) : integer;***

Mit dieser Funktion können *Alertboxen* programmiert werden. *titl* enthält dabei den Titel der Box, *msg* die Textzeilen und *but* die Buttons. Die einzelnen Textzeilen und Buttons werden jeweils mit '|' voneinander getrennt. Bei den Buttons können *Hotkeys* durch ein vorangestelltes Doppelkreuz definiert werden. Theoretisch können beliebig viele Textzeilen und Buttons benutzt werden, allerdings sollte man besonders mit den Buttons sparsam umgehen. '*OK*', '*Abbruch*' und '*Hilfe*' genügen in den meisten Fällen. *def* gibt den Default-Button an (beginnend mit 0). Als Rückgabe erhält man die Nummer des angewählten Buttons (0..n) oder M_CANCEL, wenn die Alertbox mit *<Esc>* abgebrochen wurde.

*Beispiel:* *ivar:=form_alert('TITEL','Zeile 1| Zeile 2',' #OK |#Abbruch| #Hilfe ',0);*

Es besteht auch die Möglichkeit, der Alertbox eine Nummer zu geben, um sie in der Hilfsfunktion leichter erkennen zu können (siehe dazu *Prozedur menu_setp()*). *byteout[4]* wird in diesem Fall als Eingabefeld "mißbraucht", der dort hineingeschriebene Wert wird später der Hilfsfunktion in

*byteout[3]* übergeben.

## 2.2 Prozedur ***form_error()***

***procedure form_error(num : integer);***

*form_error* zeigt den in *num* übergebenen *ASV*-Fehler in einer Alertbox an. Eine Beschreibung aller Fehler mit deren Nummern befindet sich in *V,7*. Wird eine Nummer größer *M_ERROR* übergeben, erscheint keine Meldung. Bei einer zu kleinen Fehlernummer erscheint die Meldung "***Unbekannter Fehler***" ("***Unknown error***"). Die kleinste verfügbare Fehlernummer kann mit *menu_inquire* erfragt werden. *asverror* übergibt die aktuelle Fehlernummer, viele Routinen rufen *form_error* allerdings automatisch auf.

*form_error* kann sowohl deutsche als auch englische Fehler-meldungen anzeigen. Die gewünschte Sprache kann mit *form_seterrl()* eingestellt werden.

## 2.3 Prozedur ***form_seterrl()***

***procedure form_seterrl(cc : integer);***

Mit dieser Prozedur kann die Sprache der *Application Services* (z.B. bei Fehlermeldungen) eingestellt werden. Mögliche Werte für *cc* sind *L_USA*, *L_FRG* und *L_GB*. Mit *menu_inquire* kann die aktive Sprache erfragt werden.

## 2.4 Funktion ***dialog***

***function dialog : integer;***

Die Funktion *dialog* bildet das Grundgerüst für Alertboxen, allerdings ist sie komplexer und flexibler. So kann z.B. der Text innerhalb einer Box frei plaziert werden, und auch die Bildschirm-position ist frei bestimmbar. In einer späteren Version der *Application Services* können mit dieser Funktion auch Radio-Buttons, alphanumerische Eingabefelder etc. programmiert werden.

Dialogboxen werden über das work[]-Feld programmiert und be-ginnen immer bei work[0]. Sie sind folgendermaßen aufgebaut:

work[0]:=chr(x)+chr(y)+chr(def)+chr(fl)+chr(b)+chr(h)*[+chr(nr)]*;

*x*[1.._scrx] und *y*[1.._scry] geben die Position der Dialogbox an. #255 bedeutet dabei, daß die Box auf der jeweiligen Achse zentriert wird. *def*[0..n] bezeichnet den Button, der beim Aufruf automatisch aktiv ist. *fl* muß z.Z. immer auf #0 gesetzt werden. *b* und *h* geben die Breite und Höhe der Box in Zeichen an. Der letzte Parameter ist optional, der in ihm übergebene Wert vereinfacht in der Hilfsfunktion das Erkennen der Box (siehe *menu_setp()*).

work[1]:='TITEL'; Der *Titel* der Dialogbox.

work[2]:=chr(tx)+chr(ty)+'Text 1'; Es folgen nun beliebig viele Textstrings.

*tx* und *ty* geben die Position der Zeichenkette relativ zur linken oberen Ecke der Box an.

work[3]:=#5#3'Text 2';

work[4]:=''; Zwischen Text und Buttons ein Leerstring.

work[5]:=' #OK '; Jetzt kommen die Buttons.

Hotkeys werden wie üblich durch ein vorangestelltes Doppelkreuz gekennzeichnet.

work[6]:='#Abbruch';

work[7]:=' #Hilfe ';

work[8]:=''; Zum Schluß zwei Leerstrings.

work[9]:='';

Der Aufruf einer Dialogbox geschieht mit *ivar:=dialog;*. Man erhält entweder die Nummer des angewählten Buttons (0..n) oder *M_CANCEL* zurück. Mögliche Fehler sind in *V,7* beschrieben.

# IV Mausverwaltung

**1. Allgemeine Mausfunktionen**

## 1.1 Prozedur *showmouse*

***procedure showmouse;***

Mit *showmouse* kann der Mauscursor auf dem Bildschirm sichtbar gemacht werden. Diese Routine hat - wie alle Mausfunktionen überhaupt - nur Wirkung, wenn auch eine Microsoft-kompatible Maus angeschlossen ist. Beim Start eines Programms wird diese Prozedur automatisch aufgerufen. Wenn vom Programm die Auflösung umge-schaltet wird, muß danach *showmouse* aufgerufen werden.

## 1.2 Prozedur *hidemouse*

***procedure hidemouse;***

Diese Routine läßt den Mauscursor vom Bildschirm verschwinden. Vor Bildschirmausgaben muß *hidemouse* aufgerufen werden, damit der neue Hintergund nicht zerstört wird. Danach muß der Mauscursor wieder sichtbar gemacht werden. Beim Verlassen eines Programms wird diese Prozedur automatisch aufgerufen.

## 1.3 Funktion ***mousex***

***function mousex : byte;***

Diese Funktion liefert die aktuelle X-Position der Maus in Zeichen zurück (bzw. 0, wenn keine Maus vorhanden ist).

## 1.4 Funktion ***mousey***

***function mousey : byte;***

*mousey* liefert die Y-Position der Maus in Zeichen zurück. Ist keine Maus angeschlossen, erhält man den Rückgabewert Null.

## 1.5 Funktion ***mouseb***

***function mouseb : byte;***

Von *mouseb* erhält man den Status der Maustasten zurück. Bit 0 bezeichnet die linke Maustaste, Bit 1 die rechte und Bit 2 die mittlere. Ein gesetztes Bit steht für eine gedrückte Taste. Für eine einfache Abfrage sind die Konstanten *M_LEFT*, *M_RIGHT* und *M_MID* definiert.

*Beispiel:* *if (mouseb and M_RIGHT)=M_RIGHT then writeln('Rechte Maustaste gedrückt.');*

## 1.6 Prozedur ***getmouse()***

***procedure getmouse(var mx,my,mb : byte);***

*getmouse* liefert in den mit *mx*,*my* und *mb* übergebenen Variablen die aktuelle Position des Mauscursors in Zeichen und den Status der Maustasten zurück. Ist keine Maus vorhanden, enthalten alle Variablen den Wert Null. Diese Prozedur ist aus Geschwindigkeits-gründen den Einzelfunktionen vorzuziehen, wenn mehrere Informati-onen gleichzeitig benötigt werden.

## 1.7 Prozedur ***setmouse()***

***procedure setmouse(mx,my : byte);***

Mit dieser Prozedur kann der Mauscursor an eine bestimmte Zeichenposition gesetzt werden. Dadurch kann eine *kontext-sensitive* Maussteuerung erreicht werden.

*Beispiel:* *setmouse(_scrx shr 1, _scry shr 1); { Mauscursor in die Bildschirmmitte }*

**2. Attributfunktionen**

## 2.1 Prozedur *defmouse()*

***procedure defmouse(smask,cmask : word);***

*defmouse* ändert das Aussehen des Mauscursors und aktiviert gleichzeitig den Softwarecursor des Maustreibers. *smask* gibt die Bildschirmmaske an, die mit den Bildschirmdaten UNDiert wird. Danach wird dieses Ergebnis durch ein exklusives ODER mit der Cursormaske *cmask* verknüpft und das Resultat in den Bildschirm-speicher zurückgeschrieben. Bit 15 gibt jeweils das Blinkbit an, die Bits 12-14 bestimmen die Hintergrundfarbe und die Bits 8-10 die Zeichenfarbe. In den Bits 0-7 steht der Zeichencode. Beim Start eines Programms wird automatisch folgender Befehl ausge-führt:

*Beispiel:* *defmouse($FFFF,$7700); { definiert einen invertierenden Cursor }*

## 2.2 Prozedur *mouse_sets()*

***procedure mouse_sets(sx,sy : integer);***

Diese Routine stellt die Empfindlichkeit der Maus ein. Dabei kann - für X und Y unabhängig voneinander - bestimmt werden, wie weit die Maus bewegt werden muß, damit sich der Cursor auf dem Bildschirm um ein Zeichen verschiebt. Je größer die übergebenen Werte sind, desto langsamer bewegt sich der Cursor. Die Werte werden in 1/200 Zoll gemessen. Beim Programmstart ist default-mäßig 8 in horizontaler und 16 in vertikaler Richtung eingestellt.

## 2.3 Prozedur *mouse_resets*

***procedure mouse_resets;***

*mouse_resets* stellt die Empfindlichkeit der Maus auf die Standardwerte 8 bzw. 16 für horizontale und vertikale Richtung ein.

## 2.4 Prozedur *mouse_sett()*

***procedure mouse_sett(dst : integer);***

Mit dieser Prozedur kann der Schwellenwert (in Mausbewegungen pro Sekunde) eingestellt werden, ab der die Mausgeschwindigkeit verdoppelt wird. Defaultmäßig beträgt dieser Wert 64. Möchte man die doppelte Geschwindigkeit abschalten, muß man den Wert 32767 übergeben.

## 2.5 Prozedur *mouse_resett*

***procedure mouse_resett;***

Diese Routine stellt den Schwellenwert, ab der die Mausge-schwindigkeit verdoppelt wird, auf den Standardwert von 64 Mausbewegungen pro Sekunde ein. Diese Routine wird automatisch beim Start eines Programms aufgerufen.

## V Allgemeine Auskunftsfunktionen

Seite &s

## 1. Funktion *kbshift()*

***function kbshift(bits : byte) : boolean;***

Mit *kbshift* kann festgestellt werden, ob bestimmte Umschalt-tasten gedrückt bzw. aktiv sind. Werden die angegebenen Tasten gedrückt, liefert die Funktion *true* zurück. Mögliche Werte für *bits* sind *RSHIFT*, *LSHIFT*, *CTRL*, *ALT*, *SCRLLOCK*, *NUMLOCK*, *CAPSLOCK* und *INS* (und alle Kombinationen).

*Beispiel:* *if kbshift(LSHIFT+RSHIFT) then writeln('Beide Shift-Tasten gedrückt.');*

## 2. Funktion *_scrx*

***function _scrx : byte;***

*_scrx* liefert die horizontale Bildschirmauflösung in Zeichen zurück.

## 3. Funktion *_scry*

***function _scry : byte;***

Von der Funktion *_scry* erhält man die vertikale Bildschirm-auflösung in Zeichen.

## 4. Funktion ***color***

***function color : boolean;***

Diese Funktion liefert *true* zurück, wenn eine Farbdarstellung möglich ist (monochrome EGA-Auflösungen etc. werden korrekt als *false* gemeldet).

## 5. Funktion ***mouse***

***function mouse : boolean;***

Diese Funktion ist *true*, wenn eine Microsoft-kompatible Maus verfügbar ist.

## 6. Funktion ***mvisible***

***function mvisible : boolean;***

*mvisible* liefert den Wert *true*, wenn der Mauscursor sichtbar ist. Diese Funktion liefert nur dann korrekte Werte, wenn das Ein- und Ausschalten des Mauscursors ausschließlich über *hidemouse* und *showmouse* erfolgt !

## 7. Funktion ***asverror*** & Fehlermeldungen

***function asverror : integer;***

Die Funktion *asverror* liefert die Nummer des letzten aufgetrete-nen Fehlers zurück. Danach wird *asverror* auf Null gesetzt, d.h. eine Mehrfachabfrage eines Fehlers ist nicht möglich. Ist die Rückgabe Null (*M_OK*), ist kein Fehler aufgetreten. Im folgenden nun eine Liste aller möglichen Fehler und deren Beschreibung:

***M_ERROR (-10)***

***(Allgemeiner Fehler, General Error)***

Dieser Fehler wird von *menu_watch* zurückgeliefert, wenn das Menü noch nicht initialisiert und gezeichnet ist.

***E_NREENT (-12)***

***(Doppelter Aufruf nicht möglich, Double function call not allowed)***

Dieser Fehler wird von *dialog*, *menu_watch*, *menu_rename()* und

*menu_pop()* zurückgeliefert, wenn versucht wurde, aus der Hilfsfunktion auf eine dieser Routinen zuzugreifen.

***E_NDRAW (-13)***

***(Menü noch nicht gezeichnet, Menu not drawn yet)***

*menu_kill* meldet hiermit, daß das Menü vor dem Löschen erst einmal gezeichnet werden muß.

***E_MOUTSCR (-14)***

***(Menü außerhalb des Bildschirms, Menu out of screen)***

Die Funktionen *dialog*, *menu_watch* und *menu_pop()* liefern diesen Fehler, wenn Teile des Menüs oder der Box außerhalb des Bildschirms liegen (sie müssen also verkleinert werden).

***E_MUSED (-15)***

***(Menü noch aktiv, Menu still used)***

Dieser Fehler wird von *menu_init* zurückgeliefert, wenn ein Menü angemeldet wird, obwohl das alte noch nicht mit *menu_kill* abgemeldet wurde.

***E_NINIT (-16)***

***(Menü nicht initialisiert, Menu not initialized)***

Bevor *menu_draw* aufgerufen wird, muß das Menü mit *menu_init* angemeldet werden, sonst erscheint diese Fehlermeldung.

***E_NBUT (-17)***

***(Dialogbox enthält keinen Button, Button missing in dialog box)***

Alert- und Dialogboxen müssen mindestes einen Button besitzen, andernfalls wird von *dialog* bzw. *form_alert()* dieser Fehler gemeldet.

***E_DSTRUC (-18)***

***(Fehler in Dialogstruktur, Error in dialog structure)***

*dialog* liefert diesen Fehler zurück, wenn die Länge von work[0] kleiner als 6 Zeichen ist oder die Länge einer Textzeile weniger als 2 Zeichen beträgt.

***E_DTBIG (-19)***

***(Dialogbox zu groß, Dialog box too big)***

*dialog* und *menu_pop()* brechen mit dieser Fehlermeldung ab, wenn die Box so groß ist, daß nicht genügend Speicherplatz zum Retten des Hintergrundes zur Verfügung steht (Dialogboxen sollten deshalb nie größer als 80*25 Zeichen sein).

***E_MSTRUC (-20)***

***(Fehler in Menüstruktur, Error in menu structure)***

*menu_add()* liefert diesen Fehler, wenn der übergebene String leer oder das work[]-Feld voll ist. Bei *menu_delete()* bedeutet dieser Fehler, daß der zu löschende Eintrag leer ist oder daß der Eintrag nicht zum letzten Menü gehört. *menu_rename()* meldet hiermit, daß ein leerer String übergeben wurde oder daß ein leerer Eintrag bzw. ein Menütitel umbenannt werden sollte.

*menu_pop()* schließlich liefert diesen Fehler zurück, wenn die Länge des ersten work[]-Eintrages kleiner als 4 ist oder wenn nur leere Einträge bzw. ein Eintrag der Länge 1 übergeben wurden.

*Beispiel:* *ivar:=asverror;*

*if ivar=M_OK then writeln('Es ist kein Fehler aufgetreten.');*

## VI Sonstiges

Seite &s

1. Prozedur ***cls()***

***procedure cls(att : byte);***

*cls* löscht den Bildschirm und färbt ihn mit dem in *att* übergebenen Farbwert ein. Ist ein Pulldown-Menü aktiv und gezeichnet, wird die Menüleiste übersprungen.

2. Funktion ***attr()***

***function attr(ink,pap : byte) : byte;***

Die Funktion *attr* berechnet aus der in *ink* übergebenen Zeichen-farbe und der Hintergrundfarbe in *pap* ein Farbbyte, das z.B. für *menu_setc()* oder *cls()* verwendet werden kann. In TurboPascal sind bereits Konstanten in der Unit *Crt* für die einzelnen Farben vor-definiert:

black = 0 { Schwarz } darkgray = 8 { Dunkelgrau }
blue = 1 { Blau } lightblue = 9 { Hellblau }
green = 2 { Grün } lightgreen = 10 { Hellgrün }
cyan = 3 { Türkis } lightcyan = 11 { Helltürkis }
red = 4 { Rot } lightred = 12 { Hellrot }
magenta = 5 { Violett } lightmagenta = 13 { Hellviolett }
brown = 6 { Braun } yellow = 14 { Gelb }
lightgray = 7 { Hellgrau } white = 15 { Weiß }

Man muß darauf achten, daß der Farbwert der Hintergrundfarbe den Wert 7 nicht übersteigt, da sonst das Blinkbit gesetzt wird.

## Anhang A: Definitionen

```
unit asv;
{$A+,B-,D-,E-,F+,I-,L-,N-,O+,R-,S-,V+}

interface

 const _magic     = 18011972;
     _asv      = 1.0;      _wmax     = 99;
     _bmax     = 14;       _imax     = 9;
     M_EXIT    = 255;
     M_OK      = 0;        M_HELP    = 254;
     M_NOP     = -1;       H_PULL    = 0;
     M_CANCEL  = -2;       H_POP     = 1;
     M_ERROR   = -10;      H_DIAL    = 2;
     E_NFOUND  = -11;      M_DIS     = 253;
     E_NREENT  = -12;
     E_NDRAW   = -13;      KEY_NONE  = 0;
     E_MOUTSCR = -14;      KEY_NORM  = 127;
     E_MUSED   = -15;      KEY_SPEC  = 27;
     E_NINIT   = -16;
     E_NBUT    = -17;      P_OFFS    = 123;
     E_DSTRUC  = -18;
     E_DTBIG   = -19;
     E_MSTRUC  = -20;
```

```
      RSHIFT    = 1;       LSHIFT    = 2;
      CTRL      = 4;       ALT       = 8;
      SCRLLOCK  = 16;      NUMLOCK   = 32;
      CAPSLOCK  = 64;      INS       = 128;
      L_USA     = 1;       PC        = 0;
      L_FRG     = 2;       XT        = 1;
      L_GB      = 8;       AT        = 3;
      A_NCHECKED = 1;      A_CHECKED  = 2;
      A_ENABLED  = 3;      A_DISABLED = 4;
      M_LEFT    = 1;       M_MID     = 4;     M_RIGHT = 2;
      K_F1      = 59;      K_F2      = 60;    K_INS   = 82;
      K_F3      = 61;      K_F4      = 62;    K_DEL   = 83;
      K_F5      = 63;      K_F6      = 64;    K_HOME  = 71;
      K_F7      = 65;      K_F8      = 66;    K_END   = 79;
      K_F9      = 67;      K_F10     = 68;    K_PGUP  = 73;
      K_ESC     = 27;      K_TAB     = 9;     K_PGDN  = 81;
      K_BS      = 8;       K_RET     = 13;    K_UP    = 72;
      K_LF      = 10;      K_FF      = 12;    K_DOWN  = 80;
                           K_LEFT    = 75;    K_RIGHT = 77;

 var work    : array [0.._wmax] of string;
     intout  : array [0.._imax] of integer;
     byteout : array [0.._bmax] of byte;
     void    : types.void;

function  kbshift(bits : byte) : boolean;
procedure showmouse;
procedure hidemouse;
function mousex : byte;
Seite &s
```

```
function mousey : byte;
function mouseb : byte;
procedure getmouse(var mx,my,mb : byte);
procedure setmouse(mx,my : byte);
procedure defmouse(smask,cmask : word);
function  attr(ink,pap : byte) : byte;
procedure cls(att : byte);
function  dialog : integer;
function  form_alert(titl,msg,but : string; def : byte) : integer;
procedure form_error(num : integer);
procedure form_seterrl(cc : integer);
procedure menu_init;
procedure menu_draw;
function  menu_watch : integer;
procedure menu_kill;
procedure menu_add(wrk : string);
procedure menu_delete(num : byte);
procedure menu_rename(num : byte; wrk : string);
function  menu_pop(wo : byte) : integer;
procedure menu_setc(was,saw,has,haw,dis,shdw : byte);
procedure menu_resetc;
procedure menu_setp(procv : types.procvar);
procedure menu_resetp;
procedure menu_attr(num,attrs : byte);
function  menu_stat(num : byte) : byte;
function  menu_text(num : byte) : string;
procedure menu_inquire;
function  _scrx : byte;
function  _scry : byte;
```

```
function  color : boolean;
function mouse : boolean;
function mvisible : boolean;
procedure mouse_sets(sx,sy : integer);
procedure mouse_resets;
procedure mouse_sett(dst : integer);
procedure mouse_resett;
function  asverror : integer;

implementation
...
```

## Anhang B: Programmierrichtlinien

Alle Hinweise zur Programmierung sind als Vorschlag zu sehen, damit die mit den *Application Services* erstellten Programme möglichst eine einheitliche Benutzerführung haben (*ASV* selbst hält sich im

wesentlichen an die ***S**tandard **A**pplication **A**rchitecture*). In einigen Fällen allerdings sind Abweichungen vom Standard sinn-voller. Man sollte dabei aber immer an eine einfache und gute Bedienbarkeit des Programms denken.

Grundsätzlich sollten alle Pulldown- und Popup-Menüs, Alert- und Dialogboxen wahlweise über die Cursortasten etc., eine Maus (beides automatisch) als auch über **Hotkeys** zu bedienen sein. Die Hotkeys sollten so sinnvoll gewählt werden, daß sie leicht zu erlernen und zu merken sind. Bei Dialog- und Alertboxen sind '*O*', '*A*' und '*H*' *immer* für '*OK*', '*Abbruch*' und '*Hilfe*' zu verwenden. Dialog- und Alertboxen (und evtl. auch Popup-Menüs) sollten nach Möglichkeit auf dem Bildschirm **zentriert** werden.

**a) Programmierung**

Die *Application Services* stellen eine Vielzahl von **Konstanten** zur Verfügung. Man sollte diese bei der Programmierung auch be-nutzen:

*form_error(**E_MOUTSCR**);  form_error(-14);*

Beide Befehle bewirken genau dasselbe. Allerdings ist nicht garan-tiert, daß in einer späteren Version der *Application Services* die Fehlermeldung *E_MOUTSCR* auch noch die Nummer *-14* besitzt (obwohl dieses recht wahrscheinlich ist). Außerdem ist ein mit Konstanten programmiertes Listing später sehr viel einfacher zu lesen. Die erste Version ist deshalb vorzuziehen.

Wenn die von *ASV* vordefinierten Arrays für eigene Zwecke ver-wendet werden, sollten - besonders bei späteren Versionen der *Application Services* - die Konstanten *_wmax*, *_imax* und *_bmax* zur Ermittlung der maximalen Größe verwendet werden. Eine minimale Größe von *99* bei *_wmax*, *9* bei *_imax* und *14* bei *_bmax* ist allerdings garantiert.

**Wichtig:** Bedenken Sie bei der Programmierung immer, daß Ihr Programm nach Möglichkeit mit 80*25 Zeichen auskommen sollte. Die unterste Zeile sollte außerdem für Hilfsmeldungen reserviert bleiben. Soll Ihr Programm auch auf Farbbildschirmen laufen, darf die Höhe und Breite einer Dialogbox etc. maximal *_scrx-1* bzw. *_scry-1* betragen (wegen des Schattens).

**b) Pulldown-Menüs**

Pulldown-Menüs sollten grundsätzlich nicht überladen werden. Wenn ein Menü einmal zu groß wird, sollten Sie weniger wichtige Funktionen bzw. Funktionen, die sich in Gruppen zusammmenfassen lassen, in Popup-Menüs auslagern.

Das **erste** und **letzte Menü** sollte immer so ähnlich wie im folgen-den beschrieben aufgebaut sein (weitere Details: *MENUDEMO.PAS*):

***#Datei*** ... ... ***#Fenster***
*#Neu...* *#Info...*
*#Laden...* *#Hilfsfunktion...*
*#Speichern* -
*Speichern #unter...* *#Einstellungen...*
*#Alles speichern* -
*S#chließen* *#1 DATEI1.EXT*
- ...
*#Betriebssystem...*
-
*#Ende*

**c) Popup-Menüs**

```
Seite &s
```

Bei Popup-Menüs braucht man nur darauf achten, daß diese nicht zu groß werden und daß sie sich an sinnvollen Stellen auf dem Bildschirm befinden (wenn oben links ein Menüeintrag angewählt wird und unten rechts das Popup-Menü erscheint, ist dies meistens weniger sinnvoll).

**d) Alertboxen**

Man sollte bei Alertboxen darauf achten, daß der Text **kurz und knapp** die wichtigsten Informationen enthält. Weitere Informationen können durch einen '***Hilfe***'-Button zur Verfügung gestellt werden. Auf jeden Fall sollte eine Alertbox einen '***OK***'- und '***Abbruch***'- bzw. einen '***Ja***'- und '***Nein***'-Button besitzen, es sei denn, die Alertbox gibt nur Informationen aus. Dann genügt ein '*OK*'-Button. Schließlich sollte auch ein **Titel** nicht vergessen werden.

**e) Dialogboxen**

Bei den Dialogboxen gibt es z.Z. nicht viele Richtlinien. Man sollte nur darauf achten, daß sie die Buttons '***OK***','***Abbruch***' und '***Hilfe***' o.ä. und einen **Titel** enthalten und daß sie an geeigneten Stellen auf dem Bildschirm erscheinen.

**f) Hilfsmeldungen**

Hilfsmeldungen von Pulldown- und Popup-Menüs sollten immer in der untersten Bildschirmzeile ganz links erscheinen. Man erreicht dies am besten mit ***gotoxy(1,_scry);***. Diese Meldungen dürfen auch *kontext-sensitiv* sein, d.h. sie ändern sich je nach Status der zugrundeliegenden Funktion.

## Anhang C: Beispielprogramm *MENUDEMO.PAS*

```
Seite &s
```

*<u>Hinweis:</u> Alle Bemerkungen aus dem Listing wurden gestrichen !*

*Zu lange Zeilen wurden sinnwahrend gekürzt.*

```
program MenuDemo;
 uses crt,asv;
```

Hier werden die Unit *Crt* (für GotoXY etc.) und die Unit ***ASV*** eingebunden.

```
 var ret,pret : integer;
    dcount   : byte;
    info     : string;
```

Die Variablendeklaration. *ret* und *pret* sind Rückgabevariablen, *dcount* wird als Zählvariable benötigt und *info* nimmt Meldungen für eine Alertbox auf.

```
{$F+} procedure hilfe(wnum : byte); {$F-}
 begin
  gotoxy(1,_scry);
  write('  [ ca. 75 Leerzeichen ]  ');
  gotoxy(1,_scry);
```

Die unterste Bildschirmzeile wurde gelöscht und der Cursor neu positioniert. Nun werden die Hilfsmeldungen ausgegeben.

```
  if wnum=M_HELP then
   begin
    write('erweiterte Hilfsfunktion (Demo): ');
```

Bei der *erweiterten Hilfsfunktion* wird die Art des Menüs in byteout[2] übergeben.

```
    case byteout[2] of
     H_PULL: write('Pulldown-Menü');
      H_POP: write('Popup-Menü   ');
     H_DIAL: write('Dialogbox    ');
    end;
    write('  (Bitte drücken Sie eine Taste)');
    void.c:=readkey;
    gotoxy(1,_scry);
```

```
    write('  [ ca. 78 Leerzeichen ]  ');
```

Die erweiterten Hilfsfunktionen sind selbst dafür verantwortlich, den Bildschirm wieder zu restaurieren.

```
  end
 else
  case wnum of
```

Zunächst folgen die Pulldown-Menü-Meldungen.

```
   1..8,27: write('Funktion nicht anwählbar ...');
        10: write('Beendet das Demo und kehrt zum Betr...');
        13: write('Alertbox-Demo');
        14: write('Gibt eine Fehlermeldung aus');
        15: write('Schaltet zwischen deutschen und eng...');
        17: write('Fügt im Menü "Fenster" einen Menüei...');
        18: write('Löscht im Menü "Fenster" den letzte...');
        20: write('Dieser Eintrag ist disabled');
```

Dieser Eintrag wird *nie* aufgerufen !

```
        21: write('Ruft ein Popup-Menü auf');
        23: if menu_stat(23)=A_CHECKED then write('Dies...')
            else write('Dieser Eintrag kann selektie...');
```

Ein kleines Beispiel für *kontext-sensitive* Programmierung.

```
        26: write('Gibt die Copyright-Meldung aus');
        29: write('Dialogbox-Demo (läuft nur mit mind....');
     31..39: write('Platzhalter für bis zu 9 geladene D...');
```

Jetzt kommen die Popup-Menü-Meldungen.

```
    P_OFFS+0: write('PopUp-Menü-Eintrag Nr.1');
    P_OFFS+2: write('PopUp-Menü-Eintrag Nr.2');
    P_OFFS+3: if work[25,1]=#0 then write('Dieser Eintrag...')
              else write('Dieser Eintrag kann deselekt...');
```

Noch eine *kontext-sensitive* Hilfsmeldung.

```
    P_OFFS+4: write('Dieser Eintrag kann nicht angewählt...');
```

Auch dieser Eintrag kann nicht aufgerufen werden.

```
    P_OFFS+6: write('Der 5. PopUp-Menü-Eintrag');
```

```
    end;
 end;


procedure work_init;
 begin
```
In dieser Prozedur wird die Menüstruktur für das Pulldown-Menü initialisiert.
```
  work[0]:='#Datei';
  work[1]:='#Neu...';
  work[2]:='#Laden...'
  work[3]:='#Speichern';
  work[4]:='Speichern #unter...';
  work[5]:='#Alles speichern';
  work[6]:='S#chließen'
  work[7]:='-';
  work[8]:='#Betriebssystem';
  work[9]:='-';
  work[10]:='#Ende';
  work[11]:='';
  work[12]:='#Extras';
  work[13]:='#Alertbox';
  work[14]:='Fehlermeldung';
```
Dieser Eintrag besitzt *keinen Hotkey*, damit er nicht aus Versehen angewählt werden kann.
```
  work[15]:='#Fehlermeldungen: Deutsch';
  work[16]:='-';
  work[17]:='Eintrag #hinzufügen';
  work[18]:='#Eintrag löschen';
  work[19]:='-';
  work[20]:='Das ist #disabled...';
  work[21]:='#PopUp-Menü...';
  work[22]:='-';
```

```
  work[23]:='Select It !';
  work[24]:='';
  work[25]:='#Fenster';
  work[26]:='#Info...';
  work[27]:='#Hilfsfunktion...';
  work[28]:='-';
  work[29]:='#Einstellungen...';
  work[30]:='-';
  work[31]:='#1 DATEI1.EXT';
  work[32]:='';
  work[33]:='';
 end;

begin
 work_init;
```

Zuerst wird die Menü*struktur* initialisiert.

```
 menu_init;
```

Danach wird das Menü zur Bearbeitung angemeldet.

```
 info:=' ASV - Application Services v1.0|Copyright (c)08.09....';
 info:=info+'  Koppelweg 16a * 3201 Diekholzen|       Tel.: ...';
```

Das ist der Text für eine Alertbox. Es folgen nun die Einträge für ein Popup-Menü.

```
 work[20]:=#255#255#2#0;
```

Das Menü erscheint *X- und Y-zentriert* auf dem Bildschirm. Zunächst ist work[24] der *Default-Eintrag*, dieser wird allerdings später geändert (*work[20,3]:=chr(...);*).

```
 work[21]:='POPUP-MENÜ';
 work[22]:=#0'Nummer #1';
```

Dieser Eintrag besitzt keine besonderen Attribute (*#0*).

```
 work[23]:='-';
 work[24]:=#0'Nummer #2';
 work[25]:=chr(A_CHECKED)+'#Select it !';
```

Hier wird das *CHECKED-Attribut* verwendet.

```
work[26]:=chr(A_DISABLED)+'Das ist #disabled...';
work[27]:='-';
work[28]:=#0'Das ist ein #Test !';
work[29]:='';
work[30]:='';
menu_setp(hilfe);
```

Pulldown- und Popup-Menüs zeigen nun über die Prozedur *hilfe* ihre Hilfsmeldungen an.

```
menu_draw;
```

Die Menüleiste wurde gezeichnet. Eintrag Nr.20 wird jetzt nicht anwählbar gemacht.

```
menu_attr(20,A_DISABLED);
dcount:=50;
lowvideo;
repeat
 repeat
  ret:=menu_watch;
```

Diese Zeile überwacht die *Menüanwahl*.

```
  case ret of
   M_NOP: ;
```

Bei Rückgabe dieses Wertes wurde das Menü nicht angewählt, es können also *andere Routinen des Programms* aufgerufen werden.

```
   13: void.i:=form_alert('ALERTBOX','Dieser Alertbox be...',
                          ' #1 | #2 | #3 |#Abbruch|',3);
```

Der Aufruf einer Alertbox kann sich auch über mehrere Zeilen erstrecken.

```
   14: form_error(M_ERROR);
   15: begin
```

Die nächsten beiden Zeilen können ersetzt werden durch "*if menu_text(15)='Fehlermeldungen: Deutsch' then*".

```
       menu_inquire;
       if intout[0]=L_FRG then
        begin
```

```
          form_seterrl(L_GB);
          menu_rename(15,'Error #messages: English');
        end
      else
        begin
          form_seterrl(L_FRG);
          menu_rename(15,'#Fehlermeldungen: Deutsch');
        end;
    end;
  17: if dcount<58 then
      begin
        menu_add('#'+chr(dcount)+' DATEI'+chr(dcount)...);
        inc(dcount);
      end
    else void.i:=form_alert('','Mehr als...','  #OK  ',0);
```

Der *Rückgabewert* der Alertbox wird nicht benötigt.

```
  18: begin
      menu_inquire;
      if intout[7]>30 then
```

In *intout[7]* steht der letzte Menüeintrag, der nach Möglichkeit gelöscht wird.

```
        begin

          menu_delete(intout[7]);
          dec(dcount);
        end
      else void.i:=form_alert('','Es sin...','  #OK  ',0);
    end;
  21: begin
      pret:=menu_pop(20);
```

Das *Popup-Menü* beginnt erst bei work[20], damit es nicht von der Dialogbox

(s.u.) gestört wird.

```
        if pret=3 then
          if work[25,1]=#0 then work[25,1]:=chr(A_CHECKED)
            else work[25,1]:=#0;
        if pret>=M_OK then work[20,3]:=chr(pret);
```

Wenn ein Eintrag angewählt wurde, wird dieser als *Default-Eintrag* übernommen.

```
      end;
    23: if menu_stat(23)=A_CHECKED then menu_attr(23,A_NCH...)
        else menu_attr(23,A_CHECKED);
    26: void.i:=form_alert('INFO',info,'  OK  ',3);
    29: begin
```

Jetzt kommt eine *Dialogbox*.

```
        work[0]:=#255#255#0#0#79#24;
```

Die Dialogox wird zentriert, der erste Button ist der Default-Button und die Box ist 79*24 Zeichen groß.

```
        work[1]:='EINSTELLUNGEN';
        work[2]:=#10#5'Das ist eine';
```

Der Versatz dieses Textes beträgt 10 Zeichen in X- und 5 Zeichen in Y-Richtung relativ zur linken oberen Ecke der Box.

```
        work[3]:=#12#7'recht große';
        work[4]:=#8#10'Dialogbox.';
        work[5]:='';
        work[6]:='  #OK  ';
        work[7]:='#Abbruch';
        work[8]:=' #Hilfe ';
        work[9]:='';
        work[10]:='';
        void.i:=dialog;
      end;
  end;
until ret=10;
```

Die Schleife wird verlassen, wenn der '*Ende*'-Eintrag angewählt wurde.

```
  ret:=form_alert('ENDE','Wollen Sie d...',' #Ja  | #Nein ',1);
until ret=0;
```

Diese Schleife wird verlassen, wenn die Alertbox mit '*Ja*' bestätigt wurde.

```
menu_kill;
```

Die Menüzeile wird gelöscht, der Hintergrund restauriert und das Menü abgemeldet.

```
end.
```