Dialog engine 1.1 **(Sharewareversion)**

Autor: **Dipl.-Phys. Uwe Janiak**
letzte Änderung: **17. Mai 1992**

Die ersten Schritte

Nachdem Sie das Installationsprogramm beendet haben, sollten sie die Dialogengine starten. Am einfachsten ist es, wenn Sie die mitgelieferten Beispiel Marker.Dlg und Rauh2.Dlg zuerst compilieren. Zu diesem Zweck gehen Sie auf 'File' um einen der beiden Dialoge auszuwählen.
Da sie sicher sehen wollen, was es mit diesen Beispielen auf sich hat, sollten Sie die Option Testprogramm ankreuzen. Anschließend betätigen Sie 'Compile' und es werden die Dateien Marker.Pas bzw. Rauh2.Pas und ein Testprogramm 'Testdlg.pas' erstellt.

Wechseln Sie nun zu Turbopascal für Windows und öffnen Sie Testdlg.Pas. Mit CRTL-F9 lassen Sie den generierten Quelltext in eine EXE-Datei umwandeln. Dieser Vorgang wird ohne Eingriffe von Ihrer Seite und ohne Compilerfehler durchgeführt. Sollte eine fehlende Resource angemahnt werden, ändern Sie in 'Options/Directories' das Resourcenverzeichnis in eine leere Angabe.

Nachdem das ablauffähige Programm vor Ihnen steht, sollten Sie den Dialog testen. Bei allen Elementen sollte Ihnen auffallen, daß diese voreingestellt sind, denn leere Listboxen sind ja so öde! Fall Sie Rauh2 als Testobjekt gewählt haben wird Ihnen schon beim Ablauf der Engine auffallen, daß satte 1500 Zeilen erzeugt werden. Wenn Sie diesen Dialog jedoch starten, werden Sie ein Gefühl bekommen, welche komplexen Objekte die Engine in Windeseile mit einem lauffähigen Gerüst versieht. Versuchen Sie doch einmal ungültige Zahlen oder sinnlose Werte einzugeben und 'Fertig' zu betätigen. Sie bekommen in jedem Fall die automatische Wertkorrektur zu spüren, so daß man sich wesentlicherem als der Verwaltung von Dialogobjekten widmen kann.

Sollten Ihnen die Aktionen bis dahin zugesagt haben, lesen Sie ein wenig in der beigelegten Hilfe, die Ihnen alle Fragen zur Verfahrensweise in Form einer Einführung und einer Referenz beantwortet. Nutzen Sie die Online-Hilfe bei der Erstellung Ihrer Dialogvorlagen und falls es Ihnen zu lästig erscheinen sollte Notepad.exe zu starten oder in der Hilfe nach korrekter Synthax zu suchen, lassen Sie sich registrieren und arbeiten Sie mit der Vollversion im Frage-Antwort-Spiel. Sollte Ihnen auch dieser Komfort nicht reichen, lassen Sie sich auch noch das Schreiben von Hauptprogrammen durch das Gesamtprogramm SOURCER abnehmen, daß auch noch mehr Funktionen bereitstellt.

Fürs erste jedoch, sollten Sie die obige Übung durchführen und sich die *.Dlg und generierten *.Pas Dateien ansehen. Anschließend sollten Sie die Hilfetexte sorgfältig lesen und eigene Experimente durchführen.

Also dann viel Spaß!

Wie entstand die Engine?

Seit meinem Physikstudium bin ich in der Entwicklung von Meßgeräten für die Mikroelektronik tätig. Da ich auch für die Entwicklung von Steuerprogrammen und Nutzeroberflächen für diese Geräte zuständig bin, habe ich ständig Gebrauch von möglichst bequemen und auch schnellen Programmiersystemen gemacht. Als im Frühjahr 1990 die Version 1.0 von Turbopascal für Windows auf den Markt kam, erschien diese mir als ideales Werkzeug, effektiv die Vorteile der grafischen Benutzeroberfläche nutzen zu können. Nach einiger Lernzeit hatte ich die ersten Schwierigkeiten überwunden und war in der Lage die mitgelieferten Programme zu modifizieren und eigene Programme zu schreiben. Es ist sicherlich kein Problem ein kleines Programm, welches vielleicht zwei oder drei Eingaben in einem Dialog vereint und ein paar Ausgaben auf den Bildschirm macht zu schreiben. In diesem Stadium arbeitete ich mit dem System 'Copy and modify'. Dies hatte solange Sinn, wie die mitgelieferten Strukturen ausreichten. Bei einer von mir erwarteten professionellen und schnellen Programmentwicklung war dies aber nicht lange der Fall. So gab es beispielsweise keinen Dialog der mehr als 50 Dialogelemente kombinierte, keine grafischen Darstellungen wurden unterstützt. Aus diesem Grunde begann ich mir zu überlegen, wie man wohl rationeller vorgehen könnte.
Die Lösung war ganz einfach die Art und Weise, wie in OOP Programme geschrieben werden. Bei Dialogen hat man meist den Wunsch eine Menge von Daten dem Bentzuer anzubieten, seine Reaktionen darauf zu kontrollieren und die Ergebnisse dieser Reaktionen wieder abzuholen. Bei der Programmierung heißt dies das Abarbeiten folgender Schritte:

1. Initialisieren des Dialogs und anmelden aller Elemente
2. Hinschaffen der Daten, die der Benutzer bearbeiten soll
3. Ablaufen der Berabeitung und prüfen der Reaktionen des Benutzers
4. Abholen und Sichern der Daten
5. Beenden des Dialogs

Wenn man schon mal unter TPW programmiert hat, weiß man, daß dies alles unter sehr großem Verwaltungsaufwand zu organisieren ist. So muß man erst einmal einen geeigneten Erben von TDlgWindow erfinden, die ganzen Methoden anmelden und schreiben, einen riesen Block von Standards für den Constructor und die Setup-Methode eintippen und hat dann schließlich einige hundert Zeilen Programm geschrieben mit dem Erfolg, daß man den im Resourceneditor entworfenen Dialog nun auch im Programm auf den Bildschirm zaubern kann. So bringt man 90% der Zeit damit zu, den Frondienst für die Windowsschnittstelle zu leisten und 10% damit zu, die eigentliche Aufgabe hinzuzufügen. Eigenlich wollte man ja nur zwei Editfelder zum Leben erwecken ...

Genau hier setzte bei mir die Idee ein, ich könnte ja einfach beschreiben, welche Elemente im Dialog vorhanden sind und schreiben mir ein Programmm, welches aus Namens- und Typangaben den Standardformalismus erzeugt. So entstand innerhalb weniger Tage die erste Version, die Standards für die gewohnten Dialogelemente bot. Aber nach kurzem Einsatz war ich wieder unzufrieden, da die Dialoge zwar recht sauber leifen, aber die Strukturierung der dargebotenen Information zu wünschen übrig ließ. Unter DOS war dies alles kein Problem, da konnte man im Textmodus eine wahre Farbenpracht verwenden. Warum war unter Windows kein Standardprogramm in der Lage, seine Dialoge auch farblich zu strukturieren? Die Lösung dieses Problems, brachte mich dazu die wm_color Meldung mit einzubauen, die Farbdefinitionen gestattet, wenn auch die Dokumentation von Turbopascal eine Hürde war, indem Sie behauptete, der Rückgabewert dieser Meldung hätte keine Bedeutung, dabei wird ein Handle auf den Hintergrundpinsel als Rückgabe erwartet. Nach einigen kleinen solcher Hürden waren die Dialoge farbtauglich.
Aber immer noch waren einige Problemen offen. So konnten keine Zahlentypen behandelt werden. Diese sollten dann innerhalb eines vorgegebenen Bereichs liegen und eine automatische Fehlerüberwachung besitzen. Deshalb mußten entsprechende Erben von TEdit her. Diese Erben sind jetzt in der mitgelieferten Unit WinTools enthalten und werden von der Engine selbsttätig eingebunden. Nach dieser Methode wurde dann auch ein erstes komplexes Objekt, die mit Rollbalken steuerbare Zahl als in der Engine verfügbares Objekt realisiert. Die Idee zu diesem kombinierten Objekt stammt aus Corel Draw!, wo man so auch Zahleneingaben mit der Maus bedienbar macht. Nach all diesen Maßnahmen war aus

einem kleinen Quelltextgenerator ein etwas größeres Programm geworden.
Da der Output des Programmes ein Quelltext war, mußte zu einem Test dieser mit Turbopascal übersetzt werden. Weil es sich hierbei um eine Unit handelte, mußte ein kleines Hilfprogramm her, welches diese ansteuerte. So entstand die Option 'Testprogramm'. Bei dieser handelt es sich um eine kleine Programmvorlage, welche von der Engine für jeden Dialog kopiert und mit den Informationen über den jeweiligen Unitnamen, den Dialogtyp und den Resourcenbezeichner ausgestattet wird. Soweit, so gut, es lassen sich ohne eine einzige Pascalzeile einzutippen komplette Programme erzeugen, die auch ohne Zutun von TPW beim compilieren akzeptiert werden (wenn die Syntax der Quelle richtig war). Es war damit ein erster Zwischenzustand erreicht, bei dem ein ablauffähiger Quelltext generiert wurde. Wenn keine Wechselbeziehung zwischen den Dialogelementen nötig ist, braucht man sich den Quelltext auch gar nicht anzusehen, da die automatische Generierung auch Tippfehler umgeht!
An dieser Stelle glaubte ich, kann man das Programm auch anderen zumuten und baute eine kleine Eingabeschnittstelle und schrieb diesen Hilfetext. Mit diesem Programm denke ich, kann man sich eine Menge Arbeit sparen und sich mehr den eigentlichen Problemen widmen. Falls sie daran gefallen finden, lassen sie sich als Anerkennung für die vielen Stunden Programmierarbeit, die ich diesem Problem gewidmet ahbe registrieren und gönnen Sie sich damit eine gegenüber dieser Version deutlich verbesserte Benutzerschnittstelle und Funktionsvielfalt.

Vor einigen Monaten kam von Borland eine verbesserte Dialogunterstützung namens Borland Custom Controls als Bestandteil vom Resource Workshop und Borland C 3.0 auf den Markt. Hier werden einige neue Datentypen angeboten, die eine erweiterte Version der Engine selbstverständlich ansteuern kann. Da dies aber den Besitz eines der obigen Programme voraussetzt, habe ich die Teilung der Versionen beibehalten und biete die Erweiterung auf diese Controls zusätzlich an. Eine Bemerkung noch zum Resource Workshop. Dieser bietet die Möglichkeit, die Resourcen als RC-Script abzulegen. So könnte man auf den Gedanken kommen, diese vom Genreator einlesen zu lassen und sich so den lästigen Umgang mit den ID's zu errsparen. Dies ist in einer folgenden Version geplant, muß aber mit den zusätzlichen Optionen der Engine (z.B. Datensicherung bei Exit-Buttons) kombiniert werden. Diese Funktion wird aber auch nicht als Sharewarevariante erhältlich sein.

Abschließend möchte ich noch auf die Vollversion SOURCER hinweisen, die sich nicht nur dem Problem Dialog widmet, sondern davon ausgeht, daß ein Programmrumpf entstehen soll. So kann man zusätzlich auch einen richtigen Quellcode für ein Hauptprogramm erzeugen, der ein Menü, die richtige Beschleunigertabelle, Klassendefinitionen verwendet und die selbstgenerierten Dialoge an die Menüs anbindet! Diese Vorteile möchte ich jedoch nicht in einer Sharewareversion einbinden nd hoffe, daß sie mit dem Test dieser Version zufrieden sind und ich von Ihnen höre.

Viel Spaß mit der DialogEngine!

Berlin, d. 6.5.1992 Ihr Autor Uwe Janiak

Registrierung und Update

Vorbemerkung

Falls Sie die Dialogmaschine als nützliches Programm entdeckt haben und die Arbeit mit der Maschine auch weiterrhin voll nutzen wollen, lassen sie sich als Nutzer registrieren, da sie nur so den vollen Funktionsumfang erhalten können.

Funktionsumfang

Dies ist die Sharewareversion der Dialogengine. Diese weist gegenüber der Standardversion folgende Begrenzungen auf:

- kein Editor, der die Quelltexte anzeigt
- keine Elemente wie Währungen, Zeiten
- keine Timerobjekte und keine automatischen Aktionen am Anfang des Dialogs
- kein Dialoggesteuertes Entwerfen von Dialogquellen und keine Fehleranalyse
- keine Borland Custom Controls (BWCC-Unterstützung)
- keine Bitmaps im Dialog
- keine Objekte TSCWord, TCurrency, TDate

Die im Hilfetext beschriebenen Funktionen sind jedoch uneingeschränkt und ohne zeitliche Begrenzung nutzbar. Beachten Sie jedoch den Gewinn für die geringe Registrierungsgebühr.

Registrierung: 55,- DM(85,- DM mit BWCC Unterstützung)

Wenn Sie sich registrieren lassen, erhalten Sie die Vollversion mit oben beschriebenem Leistungsumfang.

Vollversion SOURCER

Gegenüber dem Dialoggenerator unterstützt der SOURCER auch die Generierung von Applikationen mit automatischer Einbindung von Menüs, Units, Dialogen, Tastenbeschleunigern, Icons und der Deklaration der Standardmethoden. So erhalten Sie in wenigen Minuten komplette Programmrümpfe, das leidige Eintippen von Verwaltungsrümpfen entfällt und können Sie Sich der Programmierung ihrer Aufgabe widmen.

Preis der Vollversion: 138,- DM

Falls Ihnen die Sharewarevariante Appetit gemacht hat, senden Sie Ihre Bestellung direkt an:

Uwe Janiak, Grellstraße 63, O-1055 Berlin

Die Bezahlung erfolgt per Euroscheck, in bar oder für Firmen auf Rechnung.

Erzeugen von Testprogrammen

Die Engine erlaubt auch das Testen von Dialogen. Da ein Dialog immer zu irgendeinem Programm gehören muß kann ein Testprogramm generiert werden, welches diesen aufruft. Damit aber die zugehörige Resource auch gefunden werden kann, muß dem Testprogramm deren Name übergeben werden. Im Abschnitt [Resource] wird in folgender Struktur der Resourcenbezeichner angegeben:

```
[Resource]
...                         (Resourcendateien)
name=<Bezeichner>
```

Wenn diese Zeile in der Dlg-Datei fehlt, wird kein Testprogramm generiert. Falls das Testprogramm abgeändert werden soll, so schauen sie sich Testdlg.pdl an. Dieses bildet die Vorlage für das Testprogramm, sollte aber nicht geändert werden, da die Engine hier die notwendigen Bezeichner einsetzt.

Wenn durch die Option 'Testprogramm' (Checkbox) sowohl der Dialog, als auch das Testprogramm generiert wurden, braucht nur noch Turbopascal gestartet werden und die beiden Programmteile compiliert werden. Die von der Engine generierten Programmteile sind ohne weiteres Zutun ablauffähige Programme!

Was kann die Dialogengine?

Bei der Programmierung von Anwendungen fallen viele Standardaufgaben an. Die Schnittstelle zum Anwender wird in einem Programm in der Regel über einen Dialog geführt. Unter Turbopascal für Windows muß dafür ein Erbe TDlgWindow entworfen werden, der das in einer Resource vereinbarte Erscheinungsbild ansteuert. Genau diese Schnittstelle ist das Ziel der Engine.

Der Benutzer erstellt mit der Engine blitzschnell komplette lauffähige Quelltexte für Dialoge und kann das Verhalten des Dialogs mit einem ebenfalls generierten Pascalquelltext testen.

Die Dialogengine macht aus etwa 10 Zeilen eine TPW-Quelle von ca. 200 Zeilen, die sofort compiliert werden können.

Grundlagen

Die Engine erfordert als Voraussetzung eine Resource, die das Aussehen des Dialoges bestimmt. Der Anwender erstellt hieraus eine Dialogbeschreibung (1 Zeile pro Dialogelement), die Dialogtypen und deren ID's auflistet. Dies entspricht dem Aufwand einer in C üblichen Headerdatei.

Die Engine erzeugt dann eine Unit für TPW, die den Dialog als Resource (optional) einbindet und ansteuert. Diese Ansteuerung ist ein Erbe von TDlgWindow. Das Dialogobjekt enthält dann für jedes Dialogelement eine Reaktionsmethode. In der Unit wird außerdem eine komfortable Schnittstelle zum Datentransfer installiert, die beim Start initialisiert werden kann.

Falls der Dialog nur zur Eingabe oder Bearbeitung von Daten genutzt werden soll, ist er sofort verwendbar. Falls Wechselbeziehungen zwischen den Objekten verwendet werden sollen, können diese in die installierten Routinen eingefügt werden.

Voraussetzungen...
Scriptaufbau...
Beispiele...

Voraussetzungen

Um mit der Dialogengine arbeiten zu können ist Windows 3.0 notwendig. Da sich die Engine als Unterstützung für Turbopascal für Windows von Borland versteht, ist dieses ebenfalls ratsam. Zum effektiven Arbeiten kann Sourcer zusammen mit TPW und einem Resourceneditor (entweder das etwas langsame Whitewater Resource Toolkit aus dem Turbopascal-Paket oder der separat angebotene Resource Workshop) zur Anwendungsentwicklung unter Windows eingesetzt werden.

Scriptaufbau

Vor der Benutzung der Engine sollte der gewünschte Dialog als Resource entworfen sein. Aus dieser Resource werden folgende Informationen benötigt:

- Name der Resource, da sie ja eingebunden werden muß
- ID jedes zu steuernden Elements und dessen Typ

Zu diesen Informationen müssen im Script folgende hinzugefügt werden:

- Name der Unit die entstehen soll
- Initialisierungswerte der Elemente (optional)
- welche Knöpfe beenden den Dialog, bei welchen werden Daten gesichert
- welche Knöpfe verzweigen in welche andere Dialoge
- welche Farben sollen als Hintergrund und für Texte verwendet werden

Um diese Informationen zu sammeln, benutzt die Engine folgende Struktur:

```
unit=<Name>

[Resource]
<Name>

[DataRecord]
<Daten>

[Colors]
Background =<Farbe>
Text =<Farbe>
```

Dialogelemente

Jedes Dialogelement wird im Abschnitt [DataRecord] der Quelle für die Engine durch eine Zeile beschrieben. Jede Zeile beinhaltet einen Variablennamen, den ID-Code und die zugehörigen Initialisierungen.

Die Engine unterstützt folgende Elementtypen:

Endbuttons
Checkboxen
Radiobuttons
List- und Comboboxen
Editfelder
Rollbalken
Fließkommazahlen
Fließkommazahlen mit Rollbalken
Ganze Zahlen

TFloat und TWord

Diese Zahlentypen sind in Wintools neu definiert und stellen folgenden Erben von TEdit zur Verfügung:

```
PFloat=^TFloat;
TFloat=object(TEdit)
         value,default,minimum,maximum:extended;
         stellen,komma:word;

         constructor initresource(AParent:PWindowsObject;ID:Word;
                                  d,min,max:extended;n,m:word);
         procedure GetMinMax(var min,max:extended);
         procedure Delta(step:extended);
         procedure setval(v:extended);
         function legal(var v:extended):boolean;
         function getval(var v:extended):boolean;
        end;

PWord =^TWord;
TWord =object(TEdit)
         value,default,minimum,maximum,
         stellen:word;
         constructor initresource(AParent:PWindowsObject;ID:Word;
                                  d,min,max,n:word);
         procedure setval(v:word);
         function getval(var v:word):boolean;
        end;
```

Es sei angemerkt, daß durch die Funktionen getval eine Prüfung auf Gültigkeit durchgeführt wird und dieser Wert beim Verlassen von Dialogen ausgewertet wird.

Fließkommazahlen mit Rollbalken

Quellzeile:

```
<name> = TSCFloat   /ID  [#hor] @Start@Min@Max@St@Ko@ScMin@ScMax@Inc
```

Oft ist es sinnvoll, Fließkommazahlen geringfügig zu ändern. Da aber Zahlen nur über die Mauseingabe verfügbar sind, ist die Kombination von Zahlen mit einem Rollbalken sinnvoll. Dieser rollt dann die Zahl durch den ihr erlaubten Wertebereich.
Hierzu ergibt sich die jeweilige Position des Rollbalkens als Verhältnis der aktuellen Zahl zum zugelassenen Bereich. Damit kann die Maus solche Zahlenfelder ändern und eine deutliche Verbesserung der Bedienung wird erreicht.

Voraussetzung: Da hier zwei Dialogobjekte gebraucht werden (ein Editfeld und ein Scrollbar) werden eigentlich auch zwei ID's benötigt. Hier wird die Annahme gemacht, daß das ID des Editfeldes angegeben wird und sich das ID des Scrollbars als ID+1 ergibt! Dies ist aber sicher der Fall. da es ja das Ziel ist, ein kombiniertes Objekt zu erzeugen.

Programmtechnisch sind die beiden Objekte nur über den von der Engine generierten Quelltext verbunden, vom Typ her sind sie TFloat und TScrollbar.

Optionen:

#hor Die horizontalen und vertikalen Scrollbars unterscheiden sich in ihrer Laufrichtung. Hier wird die Korrektur angegeben, so daß horizontale Scrollbars links das Minimum und rechts das Maximum und vertikale Scrollbars unten das Minimum und oben das Maximum besitzen. Die Standardannahme ist ein vertikaler Scrollbar.

Start Dieser Wert wird beim Start des Programmes als Voreinstellung verwendet (Startup der Unit).

Min Minimal zugelassener Wert für die Zahl. Bei Fehlern ertönt ein Signal.

Max Maximal zugelassener Wert für die Zahl. Bei Fehlern ertönt ein Signal.

St Anzahl der Stellen mit denen die Zahl ausgegeben wird. Dieses wird bei der anfänglichen Ausgabe, bei Rollvorgängen und Fehlerkorrekturen benutzt.

Ko Wie St, aber Anzahl der Nachkommastellen.

ScMin Minimum des korrespondierenden Scrollbars. Durch die Wahl von Minimum und Maximum kann die Auflösung des Scrollbars gegenüber dem Zahlenbereich festgelegt werden.

ScMax Analog wie ScMin das Maximum des korrespondierenden Scrollbars.

Inc Jede Betätigung der Steuerknöpfe (sb_lineup, sb_linedown) erzeugt eine Erhöhung oder Erniedrigung des Wertes um den Betrag Inc. Zusätzlich werden die Meldungen sb_pageup und sb_pagedown verarbeitet und erhöhen bzw. erniedrigen um 10*Inc. Ebenso setzt das Verschieben des Rollbalkens (sb_thumbposition) den Wert auf eine dem Wertebereich proportionale Position.

Beispiel:

```
Spannung =TSCFloat  /107   @1.0 @-5 @5 @5 @1 @0 @100 @0.1
```

Rollbalken

Quellzeile: **<name> = TScrollbar /<ID> [@Start@Min@Max]**

Die Unterstützung von Rollbalken bezieht sich auf die Implementierung der notwendigen Variablen Minimum, Maximum undStartwert, einer Reaktionsroutine und der Sicherung der aktuellen Position beim Verlassen des Dialogs in der Prozedur SaveDataToRecord.

Beispiel:

```
Temperature = TScollbar   /107  @10 @0 @50
```

Endbuttons

Quellzeile: **<name>=TButton /<ID> [Option] [Option] ...**

Der Button muß mit Name bezeichnet sein und wird durch die dreistellige Control-ID angesprochen. Als Option ist es möglich, festzulegen ob das aktivieren dieses Knopfes den Dialog beendet, ob dabei die Daten gespeichert werden sollen oder ob ein anderer Dialog aufgerufen werden soll. Für Optionen können folgende Strings angegeben werden:

#exit

Diese Option legt fest, das durch diesen Button der Dialog verlassen werden kann. Es werden keine Daten gesichert. Dies ist beispielsweise für den Abbruchbutton ausreuchend.

#save

Durch Aktivieren des Buttins werden alle Daten in den Record gesichert. Die Kombination von #save und #exit entspricht einem normaleen OK-Button.

#link @<name>

Mit dieser Option kann ein analogen Dialog aufgerufen werden. Dabei wird angenommen, daß der Dialog genauso aufgebaut ist, wie der aufrufenden (gleiche Namens- und Initkonvention wie bei Sourcer. Dabei gibt <Name> sowohl die aufzurufende Dialogunit (ohne die Endung '.Pas'), als auch den Dialogtyp 'PName' an. Deshalb sollte der verkettete Dialog ebenfalls durch SOURCER erzeugt werden, ein Blick in den generierten Quelltext hilft jedoch auch weiter. Dies gilt auch deshalb, weil angenommen wird, das auch die zugehörige Resource mit <name> bezeichnet ist. Sind alle diese Konventionen erfüllt, wird in den angegebenen Dialog verzweigt und der aktuelle Dialog anschließend fortgesetz. Es sei noch angemerkt, daß diese Option nicht zusammen mit #exit und #save eingesetzt werden sollte, da separate Verzweigungs- und Beendigungsbuttons üblich sind.

Die Endbuttons werden in der vorliegenden Version nicht mit SetText initialisiert.

***Beispiele**:*

```
Abbruch=TButton  /100 #exit
Ende   =TButton  /101 #exit #save
Options=TButton  /102 #link @OptDlg
```

Checkboxen

Quellzeile: `<name>=TCheckBox /<ID> [Option]`

Die Checkbox wird hier als boolsche Variable eingeführt. Durch die ID wird die Resource definiert und optional durch die zusätzliche Angabe eines Anfangswertes initialisiert. Im Record wird zur Wertübergabe eine Variable 'v'+<name> verwendet. Diese kann entweder TRUE oder FALSE sein. Beim Programmstart kann die Variable im Startcode der Unit initialisiert werden. Zu diesem Zweck wird '@0' für eine deaktivierte und '@1' für eine aktivierte Checkbox als Option angehängt.

Beispiele:

```
Checker1  = TCheckBox /103
Checker2  = TCheckBox /104 @0
Checker3  = TCheckBox /105 @1
```

Radiobuttons

Quellzeile: `<name>=TRadio /<ID> #<Anzahl> [Option]`

Der Elementtyp Radiobutton ist nicht direkt verfügbar. Von der Idee der Radioknöpfe ausgehend ist ein Radiobutton allein sinnlos, denn dann wäre es eine Checkbox. Es stellt im Gegensatz dazu ein abhängiges Dialogelement dar, welches nur als Gruppe wirksam ist. Die Abhängigkeiten in einer solchen Gruppe sollen hier nicht weiter beschränkt werden. Aus diesem Grunde wird als Element immer eine Kombination mehrerer Radiobuttons mit einer Verwaltungseinheit als Element vom Typ TRadio eingeführt. Damit ist die Funktion von TGroup nachgebildet, aber durch die Quelltextebene transparenter als das Pendant in der Object Windows Library. Das Objekt wird durch <name> als Stammname bezeichnet. Daraus leitet sich dann eine Verwaltungsinstanz und die einezelnen Radiobuttons ab. Über diese wird angenommen, daß entsprechend der im kollektiven Objekt vereinten Anzahl ab einem Startwert die ID's kontinuierlich folgen. Der Parameter <anzahl> gibt die Anzahl der Radiobuttons bei 0 beginnend an. Die tatsächliche Zahl der Radiobuttons ergibt sich zu <anzahl+1>, die zugehörigen ID's zählen von <ID> bis <ID>+<anzahl>. Die Anzahl muß als zweistellige Zahl angegeben werden. Der bei der Initialisierung der Unit wählbare Radiobutton wird mit der Option '@<Wert>' angewählt. Dieser gibt die Nummer des ersten aktivierten Radiobuttons an. Der Wert im Record nimmt im Gegensatz dazu für jeden einzelenen Radiobutton ein Bit in einem Longint auf, wodurch die maximale Anzahl pro Gruppe auf 32 begrenzt ist.

Beispiele:

```
Group1 = TRadio /104 #04
Group2 = TRadio /109 #07 @4
```

Editfelder

Quellzeile: `<name> = TEdit /<ID> [Option]`

Unter Windows haben Editierkontrollen sehr viele Möglichkeiten. So ist es beispielsweise möglich, einen kompletten Editor als Dialogelement zu definieren. Dies erfordert aber entsprechend aufwendige Methoden für den Datentransfer und werden von Sourcer deshalb nur begrenzt unterstützt. Um eben das Speicherproblem zu lösen, wird die maximale Länge eines Editierfeldes auf 100 Zeichen begrenzt (dies gilt für den Transferpuffer). Wenn dies nicht ausreichen sollte, kann diese Zahl entsprechend erweitert werden. Es sollte aber beachtet werden, daß dann der Inhalt eines Editierfeldes nicht mehr als Zeichenkette angesehen und nicht mehr als String verarbeitet werden kann. Für die meisten Anwendungen dürfte diese Beschränkung kein Problem darstellen, ansonsten wird dem Programmierer wenigstens die Deklaration und Einbindung in den Dialog abgenommen. Wie auch bei allen anderen Werte enthaltenden Kontrollen kann auch ein Editierfeld beim Programmstart mit einem Wert belegt werden. Dieser wird als Option '@<Text>' angegeben. Auch hier gilt für das ID, das es eine Zeichenkette von 3 Zeichen sein und eine Ziffer darstellen muß.

Beispiele:

```
Edit1 = TEdit /102
Edit2 = TEdit /103 @Startwert für die Kontrolle
```

List- und Comboboxen

Quelltext:

```
<name> = TListBox /<ID> [Options]
<name> = TComboBox /<ID> [Options]
```

Listboxen und Comboboxen werden in der vorliegenden Version gleichermaßen wie Listenobjekte behandelt. Beide Objekte könen per Voreinstellung mit einigen Zeilen geladen und eine Vorauswahl bei Programmstart angegeben werden. Ansonsten werden Listboxen analog wie Radiobuttons behandelt und stellen eine Auswahl aus einer vorrgegebenen Menge zur Verfügung. Dies mag eine Einschränkung sein, bietet aber noch genügend Arbeitserleichterung.
Die Option steuert das Aussehen bei Programmstart. Es werden mindestens zwei Strings benötigt, die durch '@' getrennt werden. Das erste gibt das erste aktuelle Element an, die weiteren füllen die Box. Die gesamlänge ist auf 255 begrenzt.

Beispiele:

```
Liste = TListBox /102
Combo = TComboBox /103 @2@A@B@C@D@E@letzter
```

Fließkommazahlen

Quellzeile:
<name> = TFloat /<ID> @Startwert@Minimum@Maximum@Stellen@Komma

Als Erbe von Dialogelementen wird die spezielle Form von Fließkommazahlen (Datentyp extended) unterstützt. Diese werden mit einer Bereichüberwachung versehen. Darüber hinaus wird bei ungültigem Wert ein Standardwert gesetzt.
Diese Einstellungen müssen angegeben werden, da sonst die Kontrolle der Gültigkeit nicht richtig entworfen werden kann. Die letzten beiden Parameter geben das Erscheinungsbild an: Sie werden in der Konvertierung in einen String verwendet:

Str(Zahl:Stellen:Komma,String)

Die so gewonnene Zeichenkette wird im Editfeld angezeigt. Umgekehrt wird eine Zeichenkette, die keine gültige Zahl darstellt, beim Verlassen des Dialoges via Save-Methode des ExitButtons geprüft und eine Fehlermeldung ausgelöst, so daß der Dialog dann nur mit gültigen und im Bereich liegenden Zahlen verlassen werden kann!

TFloat ...

Beispiel:

```
Mess1 = TFloat /102  @1@0@2@1@3@1
```

Ganze Zahlen

Quellzeile:
<name> = TWord /<ID> @Startwert@Minimum@Maximum

Als Erbe von Dialogelementen wird die spezielle Form von ganzen Zahlen (Datentyp word) unterstützt. Diese werden mit einer Bereichüberwachung versehen. Darüber hinaus wird bei ungültigem Wert ein Standardwert gesetzt.
Diese Einstellungen müssen als Option angegeben werden!

TWord...

Beispiele:

```
Mess1 = TWord /102  @1@0@2@1
```

Farbsteuerung

Im Gegensatz zu Standarddialogen unter Windows ist es in Sourcer möglich die Meldung wmCtlColor für die Einfärbung des Dialoges zu verwenden. Der Hintergund und die Standardtextfarbe *müssen* angegeben werden. Für alle anderen Elemente ist die Farbangabe möglich, wird sonst aber auf die Standardtextfarbe gesetzt. Die zugehörigen Quellzeilen haben folgende Struktur:

```
[Colors]
Background =<Farbe>
Text       =<Farbe>
[/<ID>     =<Farbe>]
[/<ID      =<Farbe>]
[...]
<Leerzeile>
```

Die ersten drei Zeilen müssen vorhanden und in genau dieser Weise stehen. Als Farbe werden die 6 nach dem Gleichheitszeichen folgenden Zeichen ausgewertet. Diese müssen eine Hexadezimalzahl in der Form BBGGRR darstellen, die als Longint-Farbdefinition verwendet werden kann.
Nach den notwendigen Angaben über Farben folgen die freiwilligen. Jedes Dialogelement fragt vor dem Zeichnen seines Clientbereiches an, welches Textfarbe und welcher Hintergrund verwendete werden soll. Als Hintergund wird für den gesamten Dialog eine Farbe verwendet. Für die aktuelle Textfarbe (z.B. hervorheben einer Elementgruppe) kann jeweils ein Farbwert (siehe oben) angegeben werden. Da hierfür keine Kenntnis des Typs des fragenden Dialogelementes notwendig ist, wird die Selektion über das ID vorgenommen. Für dieses gelten die schon mehrfach erwähnten Schreibweisen. Es ist nicht notwendig, das einzufärbende Element zu deklarieren. Falls z. B. in der Resource eine statische Textvariable definiert ist, die fix bleiben aber blau angezeigt werden soll, so kann hier deren ID mit 'FF0000' angegeben werden.
Die Liste der ID's muß mit einer Leerezeile abgeschlossen werden.

Beispiel:

```
[Colors]
Background =C0C0C0
Text       =000000
/107       =FF0000

[End]
```

Durch die Markierung des End-Abschnittes wird der Freiraum zum nächsten Abschnitt als Kennzeichen des Endes der Farbanweisungen angegeben.

Beispiele

Bsp. Nr. 1

Folgender Code erzeugt ein größeres Radiobuttonobjekt und die passenden Endknöpfe:

```
unit=marker

[Resource]
Marker.Res
name=MARKER

[DataRecord]
Radios        =TRadio   /101 #19 @0
Abbruch       =TButton  /121 #exit
Ende          =TButton  /122 #exit #save

[Colors]
Background =C0C0C0
Text       =000000
/100       =FF0000
```

Generierter Quelltext ...

Bsp. Nr. 2

Hier ein etwas komplexeres Beispiel. Es werden kombinierte Rollbalken und Fließkommazahlen verwendet.

```
unit=sbtest

[Resource]
sbtest.res
name=SBTEST

[DataRecord]
SFloat    =TSCFloat   /101  @15 @5 @95 @4 @3 @0 @100 @1
OK        =TButton    /103 #exit #save
BSFloat   =TSCFloat   /104 #hor @1  @0 @2  @4 @2 @0 @100 @0.01
nn        =TFloat     /106  @0  @0 @7  @2 @0 @0 @100

[Colors]
Background=c0c0c0
Text      =000000

[Options]
```

Generierter Quelltext ...

Generierte Unit zu Beispiel 1

Der folgende Quelltext ist direkt von Sourcer aus der Datei Marker.Dlg erzeugt worden und soll zum Verständnis der gewählten Namenskonventionen dienen.

Code für die Engine...

```
{$N+}
Unit Marker;

interface uses wintypes,winprocs,wobjects,wintools;

CONST     {Im Dialog verwendete ID-Codes der Elemente}

          id_Marker_Radios=101;
          id_Marker_Abbruch=121;
          id_Marker_Ende=122;

TYPE     {Fuer den Dialog verwendeter Datensatz}

          TRecMarker= record
                vRadios:longint;
          end;

VAR       VRecMarker:TRecMarker;

TYPE      {Definition des verwendeten Dialogfensters}
          PMarker = ^TMarker;
          TMarker = Object(TDlgWindow)

            vrb_Radios:longint;
            rb_Radios:array[0..19] of PRadioButton;
            bu_Abbruch:PBUTTON;
            bu_Ende:PBUTTON;

            DialogBrush:HBrush;
            DialogText:longint;
            DialogColor:longint;

            {Konstruktionsmethoden}
            constructor Init(AParent:PWindowsObject;Name:PChar);
            procedure SetupWindow;virtual;
            procedure wmColor(var Msg:TMessage);virtual wm_CtlColor;

            {Reaktionsmethoden}
            procedure wmRadios_0(var Msg:TMessage); virtual
id_first+id_Marker_Radios+0;
            procedure wmRadios_1(var Msg:TMessage); virtual
id_first+id_Marker_Radios+1;
            procedure wmRadios_2(var Msg:TMessage); virtual
id_first+id_Marker_Radios+2;
            procedure wmRadios_3(var Msg:TMessage); virtual
id_first+id_Marker_Radios+3;
            procedure wmRadios_4(var Msg:TMessage); virtual
id_first+id_Marker_Radios+4;
            procedure wmRadios_5(var Msg:TMessage); virtual
id_first+id_Marker_Radios+5;
```

```
            procedure wmRadios_6(var Msg:TMessage); virtual
id_first+id_Marker_Radios+6;
            procedure wmRadios_7(var Msg:TMessage); virtual
id_first+id_Marker_Radios+7;
            procedure wmRadios_8(var Msg:TMessage); virtual
id_first+id_Marker_Radios+8;
            procedure wmRadios_9(var Msg:TMessage); virtual
id_first+id_Marker_Radios+9;
            procedure wmRadios_10(var Msg:TMessage); virtual
id_first+id_Marker_Radios+10;
            procedure wmRadios_11(var Msg:TMessage); virtual
id_first+id_Marker_Radios+11;
            procedure wmRadios_12(var Msg:TMessage); virtual
id_first+id_Marker_Radios+12;
            procedure wmRadios_13(var Msg:TMessage); virtual
id_first+id_Marker_Radios+13;
            procedure wmRadios_14(var Msg:TMessage); virtual
id_first+id_Marker_Radios+14;
            procedure wmRadios_15(var Msg:TMessage); virtual
id_first+id_Marker_Radios+15;
            procedure wmRadios_16(var Msg:TMessage); virtual
id_first+id_Marker_Radios+16;
            procedure wmRadios_17(var Msg:TMessage); virtual
id_first+id_Marker_Radios+17;
            procedure wmRadios_18(var Msg:TMessage); virtual
id_first+id_Marker_Radios+18;
            procedure wmRadios_19(var Msg:TMessage); virtual
id_first+id_Marker_Radios+19;
            procedure wmAbbruch(var Msg:TMessage); virtual
id_first+id_Marker_Abbruch;
            procedure wmEnde(var Msg:TMessage); virtual
id_first+id_Marker_Ende;

            {Lokale Methoden}
            function SaveDataToRecord:boolean;
            procedure rb_GroupRadios(n:word);

          end;

implementation

{$R MARKER.RES}

{*************************************************************************}
{*              Routinen zur Verwaltung des Dialogs                      *}
{*************************************************************************}

constructor TMarker.Init(AParent:PWindowsObject;Name:PChar);
  begin
    TDlgWindow.Init(AParent,Name);
    DialogColor:=$C0C0C0;
    DialogText:=$000000;
    DialogBrush:=CreateSolidBrush(DialogColor);
    rb_Radios[0]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+0));
    rb_Radios[1]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+1));
    rb_Radios[2]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+2));
    rb_Radios[3]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+3));
```

```
    rb_Radios[4]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+4));
    rb_Radios[5]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+5));
    rb_Radios[6]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+6));
    rb_Radios[7]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+7));
    rb_Radios[8]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+8));
    rb_Radios[9]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+9));
    rb_Radios[10]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+10));
    rb_Radios[11]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+11));
    rb_Radios[12]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+12));
    rb_Radios[13]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+13));
    rb_Radios[14]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+14));
    rb_Radios[15]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+15));
    rb_Radios[16]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+16));
    rb_Radios[17]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+17));
    rb_Radios[18]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+18));
    rb_Radios[19]:=New(PRADIOBUTTON,InitResource(@self,id_Marker_Radios+19));
    bu_Abbruch:=New(PBUTTON,InitResource(@self,id_Marker_Abbruch));
    bu_Ende:=New(PBUTTON,InitResource(@self,id_Marker_Ende));
  end;

procedure TMarker.SetupWindow;
  var s:string;j:word;
  begin
    TDlgWindow.SetupWindow;
    With VRecMarker do
      begin
        {Radiobutton GroupSetup}
        j:=0;
        While (vRadios and (longint(1) shl j)=0) and (j<=19) do inc(j);
        if j>19 then
          begin rb_GroupRadios(0);vrb_Radios:=1 end else
          begin rb_GroupRadios(j);vrb_Radios:=longint(1) shl j end;
      end;
  end;


procedure TMarker.wmColor;

  function id(code_von,code_bis:word):boolean;
    var i:word;ok:boolean;
    begin
      i:=code_von;
      repeat
        ok:=getdlgitem(hwindow,i)=msg.lparamlo;
        inc(i);
      until ok or (i>code_bis);
      id:=ok;
    end;
  var s:string;c,i:word;pw:twindow;
  begin
    SetBkColor(Msg.wParam,DialogColor);
    If ID(100,100) then SetTextColor(Msg.wParam,$FF0000) else
    SetTextColor(Msg.wParam,DialogText);
    Msg.Result:=DialogBrush;
  end;

{*********************************************************************}
```

```
{*              Reaktionsmethoden der einzelnen Dialogelemente           *}
{*************************************************************************}

procedure TMarker.rb_GroupRadios(n:word);
  var i,c,anz:word;
  begin
    i:=0;
    repeat
      if rb_Radios[i]^.GetCheck=1 then rb_Radios[i]^.SetCheck(0);
      inc(i);
    until i>19;
    rb_Radios[n]^.SetCheck(1);
    vrb_Radios:=longint(1) shl n;
  end;

procedure TMarker.wmRadios_0(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(0);
  end;

procedure TMarker.wmRadios_1(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(1);
  end;

procedure TMarker.wmRadios_2(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(2);
  end;

procedure TMarker.wmRadios_3(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(3);
  end;

procedure TMarker.wmRadios_4(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(4);
  end;

procedure TMarker.wmRadios_5(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(5);
  end;

procedure TMarker.wmRadios_6(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(6);
  end;
```

```
procedure TMarker.wmRadios_7(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(7);
  end;

procedure TMarker.wmRadios_8(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(8);
  end;

procedure TMarker.wmRadios_9(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(9);
  end;

procedure TMarker.wmRadios_10(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(10);
  end;

procedure TMarker.wmRadios_11(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(11);
  end;

procedure TMarker.wmRadios_12(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(12);
  end;

procedure TMarker.wmRadios_13(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(13);
  end;

procedure TMarker.wmRadios_14(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(14);
  end;

procedure TMarker.wmRadios_15(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(15);
  end;

procedure TMarker.wmRadios_16(var Msg:TMessage);
  var i,c,anz:word;
  begin
```

```
    rb_GroupRadios(16);
  end;

procedure TMarker.wmRadios_17(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(17);
  end;

procedure TMarker.wmRadios_18(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(18);
  end;

procedure TMarker.wmRadios_19(var Msg:TMessage);
  var i,c,anz:word;
  begin
    rb_GroupRadios(19);
  end;

procedure TMarker.wmAbbruch(var Msg:TMessage);
  var i:word;
  begin
    DeleteObject(DialogBrush);
    EndDlg(i);
  end;

procedure TMarker.wmEnde(var Msg:TMessage);
  var i:word;
  begin
    If SaveDataToRecord then exit;
    DeleteObject(DialogBrush);
    EndDlg(i);
  end;

{************************************************************************}
{*              Lokale Routinen zur Verwaltung des Dialogs              *}
{************************************************************************}

function TMarker.SaveDataToRecord:boolean;
  begin
    SaveDataToRecord:=false;
    With VRecMarker do
      begin
        vRadios:=vrb_Radios;
      end;
  end;

begin

{************************************************************************}
{*        Belegung der Dialogelemente nach dem Programmstart            *}
{************************************************************************}

    With VRecMarker do
      begin
```

```
        vRadios:=longint(1) shl 0;
      end;
end.
```

Generierte Unit zu Beispiel 2

Der folgende Quelltext ist direkt von Sourcer aus der Datei SBTEST.Dlg erzeugt worden und soll zum Verständnis der gewählten Namenskonventionen und Standardmethoden dienen.

Quellcode für die Engine...

```
{$N+}
Unit Sbtest;

interface uses wintypes,winprocs,wobjects,wintools;

CONST     {Im Dialog verwendete ID-Codes der Elemente}

          id_Sbtest_fl_SFloat=101;
          id_Sbtest_sc_SFloat=102;
          id_Sbtest_OK=103;
          id_Sbtest_fl_BSFloat=104;
          id_Sbtest_sc_BSFloat=105;
          id_Sbtest_nn=106;

TYPE     {Fuer den Dialog verwendeter Datensatz}

          TRecSbtest= record
                vSFloat:extended;
                vstepSFloat:extended;
                vsbminSFloat:integer;
                vsbmaxSFloat:integer;
                vsbposSFloat:integer;
                vBSFloat:extended;
                vstepBSFloat:extended;
                vsbminBSFloat:integer;
                vsbmaxBSFloat:integer;
                vsbposBSFloat:integer;
                vnn:extended;
          end;

VAR       VRecSbtest:TRecSbtest;

TYPE      {Definition des verwendeten Dialogfensters}
          PSbtest = ^TSbtest;
          TSbtest = Object(TDlgWindow)

            sfupdate_SFloat:boolean;
            sbmin_SFloat,sbmax_SFloat,sbpos_SFloat:integer;
            sf_stepSFloat:Extended;
            sf_SFloat:PFloat;
            sb_SFloat:PScrollbar;
            bu_OK:PBUTTON;
            sfupdate_BSFloat:boolean;
            sbmin_BSFloat,sbmax_BSFloat,sbpos_BSFloat:integer;
            sf_stepBSFloat:Extended;
            sf_BSFloat:PFloat;
            sb_BSFloat:PScrollbar;
            ex_nn:PFLOAT;

            DialogBrush:HBrush;
```

```
            DialogText:longint;
            DialogColor:longint;

            {Konstruktionsmethoden}
            constructor Init(AParent:PWindowsObject;Name:PChar);
            procedure SetupWindow;virtual;
            procedure wmColor(var Msg:TMessage);virtual wm_CtlColor;

            {Reaktionsmethoden}
            procedure wmflSFloat(var Msg:TMessage); virtual
id_first+id_Sbtest_fl_SFloat;
            procedure wmsbSFloat(var Msg:TMessage); virtual
id_first+id_Sbtest_sc_SFloat;
            procedure wmOK(var Msg:TMessage); virtual id_first+id_Sbtest_OK;
            procedure wmflBSFloat(var Msg:TMessage); virtual
id_first+id_Sbtest_fl_BSFloat;
            procedure wmsbBSFloat(var Msg:TMessage); virtual
id_first+id_Sbtest_sc_BSFloat;
            procedure wmnn(var Msg:TMessage); virtual id_first+id_Sbtest_nn;

            {Lokale Methoden}
            function SaveDataToRecord:boolean;

          end;

implementation

{$R SBTEST.RES}

{*************************************************************************}
{*              Routinen zur Verwaltung des Dialogs                      *}
{*************************************************************************}

constructor TSbtest.Init(AParent:PWindowsObject;Name:PChar);
  begin
    TDlgWindow.Init(AParent,Name);
    DialogColor:=$c0c0c0;
    DialogText:=$000000;
    DialogBrush:=CreateSolidBrush(DialogColor);

sf_SFloat:=New(PFloat,InitResource(@self,id_Sbtest_fl_SFloat,15,5,95,4,3));
    sb_SFloat:=New(PScrollBar,InitResource(@self,id_Sbtest_sc_SFloat));
    sf_stepSFloat:=1;
    sbmin_SFloat:=0;
    sbmax_SFloat:=100;
    sfupdate_SFloat:=false;
    bu_OK:=New(PBUTTON,InitResource(@self,id_Sbtest_OK));

sf_BSFloat:=New(PFloat,InitResource(@self,id_Sbtest_fl_BSFloat,1,0,2,4,2));
    sb_BSFloat:=New(PScrollBar,InitResource(@self,id_Sbtest_sc_BSFloat));
    sf_stepBSFloat:=0.01;
    sbmin_BSFloat:=0;
    sbmax_BSFloat:=100;
    sfupdate_BSFloat:=false;
    ex_nn:=New(PFLOAT,InitResource(@self,id_Sbtest_nn,0,0,7,2,0));
  end;
```

```
procedure TSbtest.SetupWindow;
  var s:string;j:word;
  begin
    TDlgWindow.SetupWindow;
    With VRecSbtest do
      begin
        sb_SFloat^.SetRange(sbmin_SFloat,sbmax_SFloat);
        sf_SFloat^.SetVal(vSFloat);
        sb_BSFloat^.SetRange(sbmin_BSFloat,sbmax_BSFloat);
        sf_BSFloat^.SetVal(vBSFloat);
        ex_nn^.SetVal(vnn);
      end;
  end;


procedure TSbtest.wmColor;

  function id(code_von,code_bis:word):boolean;
    var i:word;ok:boolean;
    begin
      i:=code_von;
      repeat
        ok:=getdlgitem(hwindow,i)=msg.lparamlo;
        inc(i);
      until ok or (i>code_bis);
      id:=ok;
    end;
  var s:string;c,i:word;pw:twindow;
  begin
    SetBkColor(Msg.wParam,DialogColor);
    SetTextColor(Msg.wParam,DialogText);
    Msg.Result:=DialogBrush;
  end;

{*********************************************************************}
{*              Reaktionsmethoden der einzelnen Dialogelemente         *}
{*********************************************************************}

procedure TSbtest.wmsbSFloat(var Msg:TMessage);
  var min,max,v:extended;
  begin
    if sfupdate_SFloat then exit;
    sfupdate_SFloat:=true;
    sf_SFloat^.GetMinMax(min,max);
    if msg.wparam<>sb_thumbtrack then
      begin
        if msg.wparam=sb_lineup then sf_SFloat^.delta(sf_stepSFloat) else
        if msg.wparam=sb_linedown then sf_SFloat^.delta(-sf_stepSFloat) else
        if msg.wparam=sb_pageup then sf_SFloat^.delta(10*sf_stepSFloat) else
        if msg.wparam=sb_pagedown then sf_SFloat^.delta(-10*sf_stepSFloat)
else
        if msg.wparam=sb_thumbposition then
           sf_SFloat^.setval((sbmax_SFloat-sb_SFloat^.getposition)
                            /(sbmax_SFloat-sbmin_SFloat)*
                             (max-min)+min);
      end;
  sf_SFloat^.legal(v);
```

```
  sb_SFloat^.setposition(sbmax_SFloat-trunc((v-min)/(max-min)
                       *(sbmax_SFloat-sbmin_SFloat)));
    sfupdate_SFloat:=false;
  end;

procedure TSbtest.wmflSFloat(var Msg:TMessage);
  var min,max,v:extended;
  begin
    if sfupdate_SFloat then exit;
    sfupdate_SFloat:=true;
    sf_SFloat^.getminmax(min,max);
    sf_SFloat^.legal(v);
    if (v>=min) and (v<=max) then
    sb_SFloat^.setposition(sbmax_SFloat-trunc((v-min)/(max-min)
                         *(sbmax_SFloat-sbmin_SFloat)))
              else messagebeep(0);
    sfupdate_SFloat:=false;
  end;

procedure TSbtest.wmOK(var Msg:TMessage);
  var i:word;
  begin
    If SaveDataToRecord then exit;
    DeleteObject(DialogBrush);
    EndDlg(i);
  end;

procedure TSbtest.wmsbBSFloat(var Msg:TMessage);
  var min,max,v:extended;
  begin
    if sfupdate_BSFloat then exit;
    sfupdate_BSFloat:=true;
    sf_BSFloat^.GetMinMax(min,max);
    if msg.wparam<>sb_thumbtrack then
      begin
        if msg.wparam=sb_linedown then sf_BSFloat^.delta(sf_stepBSFloat) else
        if msg.wparam=sb_lineup then sf_BSFloat^.delta(-sf_stepBSFloat) else
        if msg.wparam=sb_pagedown then sf_BSFloat^.delta(10*sf_stepBSFloat)
else
        if msg.wparam=sb_pageup then sf_BSFloat^.delta(-10*sf_stepBSFloat)
else
        if msg.wparam=sb_thumbposition then
           sf_BSFloat^.setval((sb_BSFloat^.getposition-sbmin_BSFloat)
                           /(sbmax_BSFloat-sbmin_BSFloat)*
                            (max-min)+min);
      end;
  sf_BSFloat^.legal(v);
  sb_BSFloat^.setposition(sbmin_BSFloat+trunc((v-min)/(max-min)
                       *(sbmax_BSFloat-sbmin_BSFloat)));
    sfupdate_BSFloat:=false;
  end;

procedure TSbtest.wmflBSFloat(var Msg:TMessage);
  var min,max,v:extended;
  begin
    if sfupdate_BSFloat then exit;
    sfupdate_BSFloat:=true;
```

```
    sf_BSFloat^.getminmax(min,max);
    sf_BSFloat^.legal(v);
    if (v>=min) and (v<=max) then
    sb_BSFloat^.setposition(sbmin_BSFloat+trunc((v-min)/(max-min)
                         *(sbmax_BSFloat-sbmin_BSFloat)))
              else messagebeep(0);
    sfupdate_BSFloat:=false;
  end;

procedure TSbtest.wmnn(var Msg:TMessage);
  var min,max,v:extended;
  begin
    ex_nn^.GetMinMax(min,max);
    ex_nn^.legal(v);
  end;
{*************************************************************************}
{*              Lokale Routinen zur Verwaltung des Dialogs               *}
{*************************************************************************}

function TSbtest.SaveDataToRecord:boolean;
  begin
    SaveDataToRecord:=false;
    With VRecSbtest do
      begin
        vsbposSFloat:=sb_SFloat^.GetPosition;
        vsbminSFloat:=sbmin_SFloat;
        vsbmaxSFloat:=sbmax_SFloat;
        If sf_SFloat^.GetVal(vSFloat) then SaveDataToRecord:=true;
        vsbposBSFloat:=sb_BSFloat^.GetPosition;
        vsbminBSFloat:=sbmin_BSFloat;
        vsbmaxBSFloat:=sbmax_BSFloat;
        If sf_BSFloat^.GetVal(vBSFloat) then SaveDataToRecord:=true;
        If ex_nn^.GetVal(vnn) then SaveDataToRecord:=true;
      end;
  end;

begin

{*************************************************************************}
{*        Belegung der Dialogelemente nach dem Programmstart             *}
{*************************************************************************}

    With VRecSbtest do
      begin
        vSFloat:=15 ;
        vBSFloat:=1  ;
        vnn:=1;
      end;
end.
```