

DIALOG ENGINE 2.0

Autor: Dipl.-Phys. Uwe Richter
letzte Änderung: 17.3.1993

Inhalt

Einleitung

Die ersten Schritte

Bedienung der ENGINE

Grundlagen

Einlesen von Ressourcen

Dialogelemente

Farbsteuerung

Fontmechanismus

Verwendung von Timern

Hauptprogramme

Autorenangaben

EinstellungenUnit Wintools

Garantie

Einleitung

Wie entstand die Engine und wozu braucht man Sie?

Was kann die Engine?

Was soll demnächst kommen?

Wie entstand die ENGINE und wozu braucht man sie?

Als im Frühjahr 1990 die Version 1.0 von Turbopascal für Windows auf den Markt kam, erschien diese als ideales Werkzeug, effektiv die Vorteile der grafischen Benutzeroberfläche nutzen zu können. Nach einiger Lernzeit sind ersten Schwierigkeiten überwunden und man ist 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 arbeitet man dann mit dem System 'Copy and modify'. Dies hatte solange Sinn, wie die mitgelieferten Strukturen ausreichten. Bei einer professionellen und schnellen Programmentwicklung ist dies in der Regel 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 beginnt man sich 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 Benutzer 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 Bearbeitung 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. Eigentlich wollte man ja nur zwei Editfelder zum Leben erwecken.

Genau hier setzt die Idee an, man könnte ja einfach beschreiben, welche Elemente im Dialog vorhanden sind und ein Programm schreiben, 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 dann wieder unzufrieden, da die Dialoge zwar recht sauber liefen, 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_CtlColor` 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 Hintergrundpinself als Rückgabe erwartet. Nach einigen kleinen solcher Hürden waren die Dialoge farbtauglich.

Aber immer noch waren einige Probleme 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 Hilfsprogramm 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 Ressourcenbezeichner 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 eine Online-Dokumentation. Diese Version ging dann auch Juni diesen Jahres in den Sharewarevertrieb. Obwohl eigentlich schon recht leistungsfähig, ließ das Programm doch einigen Komfort zu wünschen übrig. So war die von mir entworfene Scriptsprache zwar recht kompakt hatte aber doch ein recht kryptisches Erscheinungsbild. Zum anderen sollte das Programm doch auch einen richtigen Programmgerüst mit Menü, Icon und Beschleunigertabelle liefern. So entstand dann die Vollversion, die eine dialoggesteuerte Eingabe von Elementen zuließ. Damit waren dann die Fehler durch eine falsche Syntax ausgeschlossen, aber man mußte immer noch recht aufwendig, die verwendeten Elemente manuell eintragen und so die ID-Nummern und Elementtypen eintragen. Aus diesem Grund entschloß ich mich im Juli die Schnittstellen nach den beiden Seiten Eingabe und Ausgabe zu überarbeiten. Auf der Eingabeseite waren die Ressourcen in den Scriptcode der Engine umzuwandeln. Da in einer Resource praktisch alle Informationen für ein Programm enthalten sind, entwarf ich einen Importfilter, der es erlaubt, direkt RC-Dateien in die Engine zu importieren. Mit einer kleinen Hilfestellung bei der Ressourcendefinition ist dieses jetzt auch bei den Datentypen, die nicht im Windows-Standard enthalten sind möglich (z.B. der Fließkommazahlentyp). Auf der anderen Seite der Engine steht die Schnittstelle zu Turbopascal. Nachdem ein Quelltext erstellt ist, möchte man diesen natürlich schnell compilieren und weiterbearbeiten können. Die Engine bekam also die Möglichkeit, die von ihr generierten Quelltexte gleich im Anschluß an ihre eigene Compilierung nach Turbopascal laden zu können, dort die (Turbopascal-) Compilation anzufordern und auch Programme starten zu können, so daß die in TPW gewohnten Befehle Ctrl-F9 und F9 auch in der Engine funktionieren, nur daß diese vorher erstmal den Pascalquelltext erzeugen muß ...

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 (z.B. BorBtn, BorCheck, BorRadio) angeboten, die Vollversion der ENGINE selbstverständlich ansteuern kann (inklusive RC-Import).

Hier liegt nun abschließend die Vollversion DIALOG ENGINE vor, die sich nicht nur dem Problem Dialog widmet, sondern davon ausgeht, daß ein Programmgerüst entstehen soll. So kann man zusätzlich auch einen richtigen Quellcode für ein Hauptprogramm erzeugen, der ein Menü und die richtige Beschleunigertabelle und Klassendefinitionen verwendet. Da ich durch die Überarbeitung der Schnittstellen zu anderen Programmen die (Dialog-) Engine abgerundet habe, wird die Anbindung der generierten Dialoge an Menüpunkte nicht in diesem Programm, sondern in einem eigenen Programm erfolgen. Ich hoffe, daß Sie der mit der Engine erstmalig ermöglichten Verbindung von Ressourcen und Quelltext ebensolchen Nutzen abgewinnen können, wie es mir in meiner eigenen Praxis als Programmierer bisher möglich war.

Viel Spaß mit der DIALOG ENGINE!

Berlin, d. 20.7.1992 Ihr Autor Uwe Richter

Was kann die ENGINE ?

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 DIALOG ENGINE macht aus etwa 10 Zeilen eine TPW-Quelle von ca. 200 Zeilen, die sofort compiliert werden können. Der erste Schritt zur Erzeugung eines Dialoges, ist das Anlegen der Dialogresource. Diese Aufgabe erledigen Sie am besten mit dem Resource Workshop von Borland, da er direkt Resourcescript-Dateien (*.RC) ausgeben kann. Die Engine unterstützt in dieser Version zusätzlich zu den Dialogobjekten von Windows auch die Zahlentypen eigenen Zahlentypen TWord und TFloat. Darüber hinaus, besteht die Möglichkeit standardgemäß die Zahlentypen mit einem Rollbalken zu kombinieren. Nachdem Sie einen passende Ressource angelegt haben speichern Sie diese einmal im Format *.RES für Turbopascal und einmal im Format *.RC für den Import in die Engine ab. Anschließend können Sie die Ressource in die Engine importieren. Da diese in einem Durchgang nur einen Dialog generieren kann, wählen Sie den gewünschten aus.

Nachdem die Engine durch den Import die Elemente des Dialoges kennt, kann man den Elementen zusätzliche Eigenschaften geben. So kann man Sie individuell einfärben, man kann die Listboxen mit Standards füllen oder die Eigenschaften und Grenzwerte von Zahlenfeldern festlegen. Oder einen Timer stellen, oder die Links zu einem anderen Dialog über einen Aktionsschalter festlegen oder, oder, oder ... Der dritte Schritt besteht dann in der Compilierung. Die Engine erledigt darüber hinaus aber auch das lästige Wechseln zu Turbopascal und das Laden oder compilieren in Turbopascal selbst, so daß Sie ohne eine einzige Zeile zu tippen eine Resource in ein ablauffähiges Programm umwandeln können und ablaufen lassen können! Probieren Sie diese Eigenschaft anhand der in der mitgelieferten Ressource DEMO.RC enthaltenen Dialoge aus. Einfach importieren und Ctrl-F9 drücken, und schon werden die Pascalquelltexte erzeugt, das Programm in TPW compiliert und der Dialog gestartet. Das Sie keine zusätzlichen Eingriffe machen müssen, verdanken Sie der integrierten Automatik, die falls notwendig allen Elementen und Variablen sinnvolle Namen gibt.

Zum Abschluß dieses Kapitels noch einige Worte zu Visual Basic. Obwohl dieses Konzept ebenfalls Ressourcen und Programmtext integriert, hat man es im Gegensatz zur Engine immernoch mit Basic zu tun. Dies bedeutet, das es sich um einen Interpreter handelt, daß man ein Runtime-Modul benötigt und das man keinen direkten Zugriff zur Struktur des Programmes hat, wie dies in den prozeduralen Hochsprachen der Fall ist. Aus diesen Gründen wird BASIC wohl für immer aus der professionellen Programmierung verbannt bleiben, da dort der Zugriff auf *alle* Systemressourcen und die Fragen der Geschwindigkeit nicht nur rethorischer Art sind. Betrachten Sie die Engine im Vergleich zu Visual Basic, so werden Sie feststellen, das es um die Verbindung von Ressource und Programm geht, ohne die Einbußen in der Performance von Basic in Kauf nehmen zu müssen.

Was soll demnächst kommen ?

An dieser Stelle sei ein kurzer Ausblick für alle diejenigen erlaubt, die sich fragen, ob sie die Engine auch in Zukunft einsetzen können. In einem nächsten Schritt wird die Engine außer Dialogen auch TWindow-Objekte erzeugen können, der RC-Import ist dafür schon vorbereitet. Damit steht dann die Verwendung als direktes Hauptprogramm oder als MDI-Client zur Verfügung. An Datentypen sollen ein Währungsformat und ein Datumsformat hinzukommen. Da der Scriptcode der Engine ein von der Sprache unabhängiges Format hat, wird demnächst auch ein Treiber für Borland-C++ und ANSI C hinzukommen, so daß Sie dann frei entscheiden können, in welcher Sprache der Dialog programmiert werden soll. Falls Sie an der Frage der Portierbarkeit interessiert sind, sollten sie an Windows NT denken, welches ab Ende 92 über die verschiedenen Rechnerplattformen integrieren wird. Da jedoch die Entwicklung erst zeigen wird, ob Windows NT gegen die Vielfältigen Erscheinungsformen von UNIX ankommen oder eine Alternative bieten kann, wird die Engine auch auf UNIX portiert werden. Ein gleiches gilt für das von IBM eingeführte OS/2. Obwohl dieses Betriebssystem deutlich besser als Windows 3.x ist, hat man diesen Vorteil sehr teuer mit einem sehr schnellen Rechner, einer Festplatte ab 100 MB und einem Hauptspeicher ab 8 MB aufwärts zu bezahlen. Außerdem gibt es kaum Anwendungen, die OS/2 benötigen, so daß aus Geschwindigkeitsgründen für die meisten Anwender (nicht die Programmierer mit ihrer besseren Hardwareausstattung!) wohl doch Windows für eine gewisse Zeit die erste Wahl bleiben wird. Damit lohnt es sich, die Engine zunächst weiter für Windows zu entwickeln. Nach einem Gespräch mit Borland wurde mir mitgeteilt, daß Borland "nicht vor Ende des Jahres" seine Compiler auf OS/2 portieren wird. Da jedoch die Engine an eine Entwicklungsumgebung gebunden ist, wird diese dann im Zug der Compiler portiert werden. Betrachten Sie dieses aber nur als Anregung, da aufgrund der Präsenz von Windows in der umgebenden Kundschaft jegliche Konkurrenz durch OS/2 wohl ohne Diskussion überwinden wird, wobei der endgültige Schlag gegen OS/2 wohl durch Windows NT geführt werden wird, daß ja in gemeinsamen Zeiten von IBM und Microsoft noch OS/2 3.0 hieß!

Nach diesem kleinen Ausflug in die Welt der verschiedenen Betriebssysteme lassen Sie mich jedoch wieder zu den weiteren Entwicklungsaspekten der Engine unter Windows zurückkommen. Der Schwerpunkt der weiteren Entwicklung unter Windows wird jedoch auf dem Sektor des Hauptprogrammes liegen, da die Engine als Dialoggenerator konzipiert wurde und das Hauptprogramm nur als Treiber für den Dialog gedacht und das Konzept der Engine bei mehr Gewicht auf diesem Punkt unübersichtlich werden würde. Ich möchte Ihnen lieber ein schnelles und praktisches Werkzeug zur Verfügung stellen und ein zweites Programm eigens für die Generierung der Hauptprogramme neben die Engine stellen. Dieses wird einen Window-Manager bekommen, der den Entwurf von Standards wie Nebenfenstern und Toolbars beinhaltet. Außerdem wird dieser die Anbindung von Dialogen und Fensterobjekten an diese Oberfläche unterstützen.

Falls Ihre praktische Arbeit mit der Engine oder als Programmierer Ihnen Ideen oder Probleme bietet, die die Engine verbessern könnten ist der Autor gerne bereit, diese in zukünftigen Versionen zu berücksichtigen. Beachten Sie, daß eine Rückkopplung zum Autor Ihnen zusätzliche Arbeit abnehmen könnte. Da ich leider ziemlich ausgelastet bin, senden Sie Ihre Vorstellungen schriftlich und präzise ein, da sonst die Sammlung und Sichtung erschwert ist. Im voraus schon vielen Dank für Ihre Mühe!

Die ersten Schritte

Nachdem Sie das Installationsprogramm beendet haben, sollten sie die DIALOG ENGINE starten. Am einfachsten ist es, wenn Sie die mitgelieferten Beispiele *.Dlg compilieren und sich das Ergebnis anschauen. Das der vorher beschriebene Import von Ressourcen diese Dialoge überschreiben würde, heben Sie sich diese Übung noch ein wenig auf. Um einen dieser Dialoge zu laden, gehen Sie auf den zugehörigen Menüpunkt im Menü DATEI oder verwenden Sie die von Turbopascal 6 gewohnte Kombination F3.

Da sie sicher sehen wollen, was es mit diesen Beispielen auf sich hat, sollten Sie die Optionen Testprogramm und Dialogerzeugung im Dialog Hauptprogramm/Vorgaben ankreuzen. Anschließend betätigen Sie 'Compile' und es werden die Dateien *.Pas für den Dialog und das Hauptprogramm erstellt, diese nach TPW geladen und dort compiliert, so daß Sie nach erfolgreicher Operation am Abschluß eines Compilervorganges in TPW sind. Damit diese Verbindung reibungslos funktioniert, sind in Turbopascal einige Vorarbeiten notwendig. Zum ersten muß die IDE gestartet sein, da die Engine TPW nicht selbst lädt. Außerdem muß in der Option Verzeichnisse ein Pfad auf die Engine gesetzt sein, oder die Unit WinTools.Tpu in einen Standardpfad kopiert werden, da die Dialoge zur Funktion diese Unit benötigen und sonst ein Compilierfehler von TPW entsteht. All letztes sollte keine Hauptdatei definiert sein, da TPW sonst die Hauptdatei statt der zuletzt geladenen Datei übersetzt.

Nachdem das ablauffähige Programm vor Ihnen steht, sollten Sie den Dialog testen. Bei allen Elementen sollte Ihnen auffallen, daß diese voreingestellt sind, denn leere Zahlenfelder sind ja so öde! Beachten Sie auch daß aus den wenigen Zeilen Scriptcode (etwa 2 pro Element) leicht mehrere hundert Pascalzeilen generiert werden! Wenn Sie die Dialoge 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 wesentlichem als der Verwaltung von Dialogobjekten widmen kann.

Sollten Ihnen die Aktionen bis dahin zugesagt haben, lesen Sie ruhig in dieser Dokumentation weiter, da Sie sonst einige Funktionen ohne Erfolg benutzen könnten. Nutzen Sie die Online-Hilfe bei der Erstellung Ihrer Dialogvorlagen. Fürs erste jedoch, sollten Sie die obige Übung durchführen und sich die *.Dlg und generierten *.Pas Dateien ansehen.

Nachdem Sie die ersten Dialoge zum Laufen gebracht haben, testen Sie die Möglichkeiten der Farbsteuerung. Rufen Sie diesen Dialog auf und ändern Sie die Hintergrundfarbe. Bei den Standarddialogen von Windows werden Sie die Wirkung sehen, bei Benutzung der Borland Custom Controls werden nur wenige Elemente mit den eigenen Farben darstellbar (die Klasse BorDlg gibt nicht alle Meldungen an die Dialogobjekte weiter und reagiert teilweise auch etwas anders als die Standardfunktionen in Fenstern).

Als letzte Übung sollten Sie den Import testen. Wählen Sie die Datei DEMO.RC über das ganz linke Icon und die Dateiauswahl aus. Anschließend wird diese RC-Datei nach Dialogen durchsucht und diese zur Auswahl angeboten. Wählen Sie sich daraus einen aus. Die Engine bestimmt die Dialogobjekte in diesem Dialog und belegt alle mit einer Standardeinstellung, lädt die Informationen über Menüs, Beschleunigertabelle und Icon (für das Hauptprogramm) und erzeugt daraus eine Datei *.Dlg im Scriptformat der Engine. Nun können Sie wie gehabt den Dialog zum Quelltext machen oder vorher die Elemente spezifizieren. Als letztes werden Sie sicher das Icon für Compilation betätigen um die neue Quelle in TPW laufen lassen zu können.

Bedienung der ENGINE

Die DIALOG ENGINE erscheint nach dem Installieren als Symbol in einer der Windows-Anwendungen und wird wie jedes Windows-Programm durch Doppelklick gestartet und durch ALT+F4 in üblicher Weise wieder geschlossen. Nach dem Starten des Programmes erscheint auf dem Bildschirm zunächst das Titelbild und anschließend das Hauptmenü .

Windows-Programme enthalten gewöhnlich eine Menüleiste mit den Auswahlmöglichkeiten, aus denen der Nutzer wählen kann. Jeder Menüpunkt kann grundsätzlich eine zusätzliche Auswahlliste anzeigen. Die meisten Windows-Programme bieten beispielweise ein Dateimenü mit den Kommandos zum Anlegen einer neuen Datei oder zum Öffnen einer bestehenden Datei.

Selbstverständlich stehen für die einzelnen Anwendungen auch entsprechende Tastenkürzel zur Verfügung, um in einer bestimmten Anwendung eine bestimmte Aufgabe auszuführen. Die Tastenkürzel können, wie bei allen Windows-Programmen grundsätzlich parallel zu den Menüfunktionen benutzt werden. Neben der Menüanwahl der Anwendungspunkte der Hauptmenüleiste oder den Tastenkürzeln ist eine Schnellauswahl noch über die darunterliegende Iconliste möglich. Hier sind die wichtigsten Anwendungen in der Form eines kleinen Bitmaps untergebracht. Die wichtigsten Funktionen, die auch unter dem Hauptmenü zu finden sind, sind durch die ersten vier Icons repräsentiert und hier auch entsprechend farblich grau im Untergrund zu den anderen Icon-Funktionen abgesetzt. Hierbei handelt es sich um die Funktionen IMPORT/LADEN, EDITIEREN, SICHERN und COMPILIEREN. Unter LADEN und SICHERN verbergen sich die zugehörigen Funktionen zum Laden bzw. Sichern der ENGINE-Quelltexte, während Import das Filtern von Ressourcscripts aktiviert . Beim Betätigen dieser Anwendungen öffnet sich wieder ein Fenster, in die die Eingabe des Namens vorgenommen oder die Auswahl unter vorhandenen Dateien in einer Listbox getroffen werden kann. Das ICON Laden/Import hat eine Doppelfunktion: ein linker Mausklick öffnet die Dateiauswahl des Imports (zwei Dialogfenster klappen auf) und ein rechter Mausklick öffnet das Laden von Enginedateien.

Unter dem Icon für COMPILIEREN wird beim Betätigen das Erzeugen eines ENGINE-Quelltextes, die sofortige Übersetzung in einen Pascal-Quelltext und dessen Kompilierung in TPW zu einer EXE.-Datei in einem Arbeitsablauf verstanden. Will man dieses Verfahren schrittweise abarbeiten, so verbergen sich unter dem Hauptmenüpunkt COMPILIEREN eine Reihe von Funktionen, die dies erlauben. So kann die Übersetzung zum Pascal-Quelltext ohne sofortige Weiterführung in Maschinensprache vorgenommen werden. Dies schließt die Möglichkeit der Kontrolle sowie manueller Änderungen ein, wobei jedoch angemerkt werden muß, daß das automatische Erzeugen der Quelltexte Schreibfehler von vorn herein ausschließt und somit auch die Compilierarbeit und Fehlersuche entschieden abkürzt.

Unter dem Icon für EDITIEREN wird einfach der zugehörige ENGINE-Quelltext eines Dialoges nach Turbospascal transportiert und kann entsprechend gelesen werden. Vor Veränderungen an diesem Quelltext sei gewarnt, da die Engine eine exakte Synthax voraussetzt und nicht alle Fehler abfangen kann. Bevor Sie diese Funktion nutzen, sollten Sie sich mit der Befehlsstruktur gründlich vertraut machen!

Grundlagen

Dialogfenster bieten dem Nutzer die Möglichkeit, mit einer Anwendung zu kommunizieren. Ein Dialogfenster erlaubt dem Benutzer, Informationen einzugeben. Hierbei handelt es sich beispielsweise um die Namen von zu eröffnenden Dateien, zu suchendem Text oder die Farbe, in der eine Aufgabe darzustellen ist.

Ein Dialogfenster enthält gewöhnlich eine Anzahl von Elementen, wie beispielsweise Schalter, Textfelder und Bildaufbauleisten. Die Informationen können mittels dieser Dialogelemente eingegeben werden. Eine Anzeige von statischem Text oder Grafiken ist in einem Dialogfenster ebenfalls möglich. Dialogelemente bieten dem Nutzer oft einfache Verfahren, eine Auswahl zu treffen. Der Benutzer kann beispielsweise durch Anklicken von Schaltern, Einschalten von Feldern oder Auswahl von Einträgen aus einer Liste Funktionen ein- und ausschalten oder entscheiden, was das Programm als nächstes tun soll.

Für den Programmierer ist das Dialogfenster ein übergeordnetes Fenster und jedes Element stellt ein untergeordnetes Fenster dar, das Botschaften entgegennimmt. Zum Erstellen eines Dialogfensters füllen Sie ein leeres Dialogfenster mit den gewünschten Elementen. Jedes Dialogelement ist für Maus- und Tastaturmeldungen verantwortlich, je nach den Eingaben, die der Nutzer vornimmt. Mit der DIALOG ENGINE wird das Erstellen von Dialogen in Turbopascal für Windows einfach möglich. In dem folgenden Kapiteln werden die Grundlagen der einzelnen Ansteuerungen sowie die Realisierung durch die Engine erklärt werden.

Was sind Windows-Ressourcen ...
Arbeitsweise der Engine ...

Was sind Windows-Ressourcen ?

Ressourcen sind Daten, die die sichtbaren Teile eines Windows-Programmes definieren. Sie als Anwender arbeiten meist dann mit den Ressourcen eines Programmes, wenn Sie beispielsweise ein Dialogfenster öffnen und durch Anklicken eines Aktionsschalters das Programm eine Aufgabe ausführen lassen.

Zusätzlich zu Dialogfenstern und Aktionsschaltern können Sie in Ihren Windows-Programmen auch andere Arten von Ressourcen verwenden, wie zum Beispiel Symbole, Mauszeiger, Bitmaps, Menüs und Tastenkürzel.

Bei Windows-Anwendungen schaffen die Ressourcen eine durchgängige Benutzerschnittstelle, die dem Anwender den Wechsel zwischen Windows-Programmen erleichtert. Das gesamte Dialogfenster und alle darin enthaltenen Elemente (Aktionsschalter, Auswahlfenster u.s.w.) sind im Windows-Programm als Ressourcen definiert.

Genau hier ist der Punkt, wo die ENGINE einsetzt. Die Ressourcen werden als RC-Datei eingelesen und von der Engine interpretiert. Dabei werden die Informationen über die Elemente des Dialoges und die enthaltenen Menüs, Icons und Beschleunigertabellen gefiltert. Diese Bezeichner werden zum Ablauf des erstellten Turbopascalprogrammes benötigt, so daß die Engine diese Informationen in den Quelltext eingehen läßt. Um aber auch darauf Zugriff zu gestatten und auch diejenigen Nutzer zu gewinnen, die keine RC-Datei bereitstellen zu können, können Sie in einem Dialog die Liste der beteiligten Ressourcendateien und den Dialogbezeichner editieren. Dieser Ressourcendialog betrifft alle Informationen, die für den Ablauf des Dialogs notwendig sind, nicht aber die für das Hauptprogramm. Für das Hauptprogramm können Sie den Dialog Vorgaben verwenden, der aber nicht an dieser Stelle erklärt werden soll.

Wenn das Menü Bearbeiten/Ressourcen angewählt wurde, öffnet sich ein Fenster. Man kann eine Ressource als Dateiverzeichnis zu einem Projekt hinzufügen oder direkt in dieses Projekt aufnehmen. An dieser Stelle wird die Deklaration bzw. Anmeldung der zur Verfügung stehenden Ressourcendateien mit ihren Dialogelementen vorgenommen. Es wird als erste Funktion der Bezeichner des Dialoges festgesetzt. Mit dem Befehl HINZUFÜGEN können Sie die in einer externen Datei gespeicherte Ressource hinzufügen bzw. mit dem Befehl LÖSCHEN oder ALLE LÖSCHEN wieder entfernen. Externe Ressourcendateien enthalten die einzelnen Dialogelemente und werden auf diese Art und Weise dem jeweiligen Projekt als zur Verfügung stehend angemeldet. Die vorhandenen Ressourcendateien sind in der rechten Listbox aufgelistet. Die aktuelle Ressourcendatei ist in dem Editfenster unter RESOURCENDATEI angezeigt.

Unter Bezeichner ist der Name der Dialogresource anzugeben, wie er im Aufruf von ExecDialog('...') benötigt wird. In der Liste werden die in den Dialog einzubindenden Ressourcendateien *.res angegeben. Dies ist von Bedeutung, wenn z.B. Bitmaps aus separaten Dateien eingebunden werden sollen.

[illegible]

```
PrimaryDelay=1000  
TimerQuit=0  
TimerSaveData=0  
TimerSignal=0
```

```
[Data Section]                                ;Datenbereich für Listboxen, Menüs etc.  
...
```

Einlesen von Resource Scripts

Ressourcen werden mit dem Kommando Alt-R bzw. dem Menüpunkt 'RC-Import' eingelesen. Als Dateiformat wird das textorientierte RC-Format (Ressource Script) erwartet, wie man es mit dem Ressource Workshop erzeugen kann, oder wie diese auch in der meisten Veröffentlichungen über Ressourcen angegeben werden. Wenn eine RC-Datei eingelesen wird, filtert die Engine alle notwendigen Informationen über Dialoge, Menüs, Beschleunigertabellen etc. heraus und kreiert ihren eigenen Zwischencode. Da aber die Engine mehr Elemente unterstützt als die Standardelemente von Windows gestatten, sind einige Regeln zu beachten, damit die Engine den Typ der Elemente richtig zuordnen kann.

Ressourcendatei auswählen

Die Importfunktion wird über das RC-Icon, Alt-R oder den Menüeintrag im Menü Datei gestartet. Anschließend öffnen sich zwei Dialoge, der oberste fordert zur Dateiauswahl auf. Nachdem diese ausgewählt ist, wird die RC-Datei eingelesen und auf vorhandene Dialoge durchsucht. Anschließend werden diese in der Listbox angeboten. Da die Engine in einem Durchgang nur einen Dialog in eine Unit umwandelt, ist dieser hier zu spezifizieren. Aus dem Ressourcenbezeichner wird durch Kürzen auf maximal 8 Zeichen ein Dateiname gewonnen, der als *.Dlg die Engine-Datei und als *.Pas die nachfolgend generierte Pascaldatei bezeichnet. Dieser Name wird natürlich nur als Vorgabe verwendet und kann vor dem compilieren geändert werden. Wenn die Auswahl des Dialoges erfolgt ist, wird sofort die entsprechende Engine-Datei generiert (der Mauscursor wird während dieser Zeit zur Sanduhr).

Regeln für die Interpretation der Ressource

Da die Engine vor dem Problem steht, den einen Typ Editfeld den Typen TEdit, TWord, TFloat, TSCWord und TSCFloat zuzuordnen, kann diese Information in dem Text des Editfeldes untergebracht werden. Dazu wird einfach der gewünschte Typ in Großbuchstaben eingetragen. Bei den Zahlentypen mit Rollbalken wird zusätzlich gefordert, das die nächste Zeile im Script den zugehörigen Rollbalken definiert und dessen ID genau um eins größer als der des Editfeldes ist. Ein ähnliches Problem liegt bei Radiobuttons vor. Diese werden von der Engine automatisch zu einer Gruppe zusammengefaßt und dürfen deshalb nicht als AutoRadioButton deklariert sein (der Stil BS_RADIOBUTTON wird zur Identifikation genutzt). Alle aufeinanderfolgenden Zeilen von Radiobuttons werden zu einer Gruppe zusammengefaßt. Dabei wird angenommen, daß die ID's auch aufeinander folgen. Falls Sie zwei Gruppen trennen wollen fügen Sie z.B. ein leeres "Static" ein.

Für die praktische Arbeit mit dem Workshop gehen Sie folgendermaßen vor. Zuerst entwerfen Sie Ihre Ressource wie gewohnt. Danach legen Sie die Reihenfolge fest. Anschließend bearbeiten Sie Ihre Ressource als Text und setzen die ID's in fortlaufender Folge. Prüfen Sie nun die Gruppen von Radiobuttons und die Gruppen von Edit und Scrollbar für Zahlen mit Rollbalken und tragen Sie die richtigen Typen in den Text der Editfelder ein. Falls Sie in diesen eine Voreinstellung wünschen, geben Sie bei der Arbeit in der Engine eine Starttext für das entsprechende Element ein. Wenn Sie importieren sollte im Menüpunkt Optionen/Einstellungen die Flagge, ob statische Elemente auch als Objekte eingelesen werden sollten beachtet werden. Namensgebung: Während des Einlesens der Resource muß an alle gefundenen Objekten ein eindeutiger und von Turbopascal akzeptierter Name vergeben werden. Die Engine arbeitet dabei in mehreren Stufen. Zuerst versucht sie aus dem Text des Objektes (bei Buttons der Fenstertext, im allgemeinen der erste String in der CONTROL-Anweisung des Resourcescripts) einen Namen von max 15 Zeichen zu gewinnen. Dabei ist es unerheblich, ob an erster Stelle eine Zahl steht, da alle Objekte einen typischen Präfix zugeordnet bekommen (z.B. "ed_" für ein Editfeld). Ebenso werden ungültige Zeichen eliminiert (nur alphanumerische sind zugelassen, deutsche Umlaute werden entsprechend umgesetzt). Falls nach dieser Regelung kein Name übrigbleibt, wird das Objekt als Element_x (x eine lfd. Nummer) bezeichnet. In der zweiten Stufe werden dann die Mehrdeutigkeiten der Namen beseitigt ("Dopx_" wird als zusätzlicher Präfix mit laufender Nummer angefügt). Dadurch ist für die meisten Fälle schon automatisch ein sinnvoller Name vergeben, ansonsten können Sie diesen in der Engine selbst ändern.

Anmerkung: Die Borland Custom Controls werden unterstützt. Dabei werden die Klassen BorBtn, BorCheck und BorRadio in die Windows eigenen Kontrollen übersetzt (natürlich nur was den Quelltext für deren Ansteuerung betrifft). Dabei genießen die Buttons mit den ID's 1,2 und 998 eine Sonderbehandlung. Falls eines dieser Elemente gefunden wird, installiert die Engine keine Antwortmethoden für die einzelnen Knöpfe sondern überschreibt die OK und Cancel-Methode von TDlgWindow und legt eine Routine zur Hilfebearbeitung an. Ebenso werden die Voreinstellungen von Buttons mit den Texten Ok und Quit als Verlassen des Dialogs mit Datensicherung und die mit den Texten Abbruch oder Cancel als Abbruch ohne Datensicherung interpretiert. Wenn Sie diese Einstellungen rückgängig machen wollen so kreuzen Sie diese einfach im Bearbeitungsdialog des Elementes ab.

Achtung: Verwenden Sie keine Auto-Checkboxes oder Auto-Radiobuttons. Da die Engine selbst die Verwaltung der Knöpfe übernimmt, müssen die Knöpfe beim Anklicken auch Meldungen abgeben. Deshalb sollten sie die Verwaltung der Engine überlassen, da sonst keine Reaktionen auf einzelne Schaltzustände möglich sind (Beispiel die Farbdialogbox der Engine selbst: Wenn Sie zwischen Vorder- und Hintergrundfarbe schalten, müssen auch die Editfelder und die Scrollbars umgesetzt werden. Erhalten Sie aber keine Meldung über das Schalten der Knöpfe, so werden auch keine Reaktionen möglich). Die Engine importiert aus diesen Gründen nur die Klasse 'BUTTON' mit den Stilen BS_RADIOBUTTON und BS_CHECKBOX als Radiobutton bzw. Checkbox. Bei den BCC werden nur die Klassen 'BorRadio' und 'BorCheck' (die Radiobuttons werden zusätzlich auf aufeinander folgende ID's geprüft und entsprechend gruppiert) berücksichtigt.

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 im einzelnen folgende Elementtypen:

*Endbuttons, Checkboxes, Radiobuttons, List- und Comboboxen, Editfelder
Rollbalken, Fließkommazahlen, Fließkommazahlen mit Rollbalken,
Ganze Zahlen, Ganze Zahlen mit Rollbalken*

Wie bereits in dem Kapitel über die Ressourcen erklärt worden ist, werden die Elementtypen über sogenannte Ressourcendateien zu dem jeweiligen Projekt hinzugefügt, d.h. dem Projekt angemeldet. Die weiter notwendigen Informationen der Dialogelemente in dem zu erstellenden Dialog werden nach dem Anmelden dann einzeln in den Dialogelement-Dialogen der DIALOG ENGINE näher spezifiziert. Diese sind unter den Element-Icons unter dem Hauptmenü oder im Hauptmenü selber unter der Funktion ELEMENT verborgen. In den jeweiligen Auswahlfenstern für das Dialogelement wird in jedem Fall immer eine Angabe über den zugehörigen ID-Code und ein Name, der das Dialogelement bezeichnet, erwartet. Beide sind für die Identifikation des Elementes essentiell und stehen am Anfang jedes Auswahlfensters. Zusätzlich erwartet werden Werte, die die Eigenschaften des Elementes näher bezeichnen. Hierzu gehören oft Anfangs- oder Startwerte, die dem Nutzer beim Aufklappen des Dialoges angeboten werden, aber nicht genutzt werden brauchen. Dies entspricht im wesentlichen einer Initialisierung. Weiterhin werden Werte für bestimmte Minima oder Maxima speziell bei Zahlen erwartet, u.s.w. ...

Das Erstellen des Dialoges macht im wesentlichen folgende Hauptarbeitsschritte notwendig:

- Erzeugen einer Dialogelemente-Ressourcendatei,
- Anmelden der zu verwendenden Ressourcendateien über die Funktion RESOURCE und
- Spezifizierung der Dialogelementeeigenschaften in den einzelnen Elemente-Auswahlfenstern.

Über die Iconleiste ist eine Schnellanwahl ohne Verwendung des Hauptmenüs möglich. Bis auf das statische Element hat man so über alle verfügbaren Elemente direkten Zugriff:

Aktionsschalter (Endbutton)
Alphanumerisches Eingabefeld
Gruppe von Radiobuttons
Markierungsfeld
Listbox
Combobox
Rollbalken
Fließkommazahl
Ganze Zahl
Fließkommazahl mit Rollbalken
Ganze Zahl mit Rollbalken
Statische Elemente

Jede Funktion eröffnet ihrerseits ein Untermenü, in dem die Eigenschaften des Dialogelementes festgelegt werden. In den weiteren Kapiteln werden neben den Erklärungen der einzelnen Dialogelemente selber auch die zugehörigen Eigenschaftendialoge vorgestellt werden.

Endbuttons

Der Endbutton ist auch ein sogenannter Aktionsschalter, der Text enthält und durch sein Betätigen ein konkretes Ereignis sofort auslöst. Das Icon Nummer 5 in der Icon-Liste bzw. die Funktion unter dem Hauptmenüpunkt ELEMENT fügt ein neues Aktionsschalter-Objekt in die Objektliste ein.

Hier wird unter der Überschrift AKTIONSSCHALTER als erstes der entsprechende ID-Code des Schalters festgelegt. Dieser ist für die korrekte Anrede des Elementes im zu erstellenden Dialog wichtig. Daneben wird der Bezeichner bzw. Name des Schalters erstellt, da dieser im zu erstellenden Programm ja irgendwie bezeichnet werden muß. Zur Funktion, die der Aktionsschalter jeweils ausübt, sind hier drei Möglichkeiten vorgegeben. Dem Betätigen des Aktionsschalters können folgende Ereignisse folgen:

1. DATEN SICHERN,
2. DIALOG BEENDEN und
3. ANDERE DIALOGS STARTEN.

Die beim Betätigen erwünschten Ereignisse sind entsprechend anzukreuzen, wobei zu beachten ist, daß beim Starten anderer Dialoge eine Verkettung erfolgt. Hierbei sind zusätzlich unter NAME und RESOURCENAME die Bezeichner der Verkettungsdialoge zu benennen. Da bei dieser Funktion davon ausgegangen wird, daß eine Subfunktion aufgerufen wird, ist die gleichzeitige Verwendung von Beenden und Verbinden ausgeschlossen.

Internes Format

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, daß durch diesen Button der Dialog verlassen werden kann. Es werden keine Daten gesichert. Dies ist beispielsweise für den Abbruchbutton ausreichend.

#save

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

#link @<name>

Mit dieser Option kann ein analoger Dialog aufgerufen werden. Dabei wird angenommen, daß der Dialog genauso aufgebaut ist, wie der aufrufende (gleiche Namens- und Initkonvention wie bei DIALOG ENGINE). Dabei gibt <Name> sowohl die aufzurufende Dialogunit (ohne die Endung '.Pas'), als auch den Dialogtyp 'PName' an. Deshalb sollte der verkettete Dialog ebenfalls durch DIALOG ENGINE erzeugt werden, ein Blick in den generierten Quelltext hilft jedoch auch weiter. Dies gilt auch deshalb, weil angenommen wird, daß auch die zugehörige Ressource mit <name> bezeichnet ist. Sind alle diese Konventionen erfüllt, wird in den angegebenen Dialog verzweigt und der aktuelle Dialog anschließend fortgesetzt. 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
```

Details zum generierten Quellcode

Die folgende Methode beinhaltet die Reaktion bei den Optionen 'Daten sichern' und 'Beenden':

```
procedure TXXX.wmBEISPIEL(var Msg:TMessage);
var i:word;
begin
  vrecXXX.ExitID:=ID_XXX_BEISPIEL;
  If SaveDataToRecord then exit;
  DeleteColorElements;
  EndDlg(i);
end;
```

Der allgemeine Aufruf SaveDataToRecord gibt den Fehler beim Abrufen der Werte der Dialogobjekte zurück. Falls also z.B. eine ungültige Zahl eingegeben wurde, wird der Dialog nicht beendet (die Zahlenobjekte der Engine besitzen eine eigene Korrekturmethode, so daß bei erneutem Betätigen der Dialog doch verlassen wird). Falls Sie spezielle Probleme haben, sollten Sie die Methode SaveDataToRecord modifizieren! Die Methode DeleteColorObjects sollte für Sie zum Vernichten aller temporären Dialogobjekte genutzt werden (die Engine benutzt zur Zeit nur Pinselobjekte und deshalb der Name) und ihre Antipode CreateColorElements zu deren Erzeugung. Falls ein anderer Dialog gestartet werden soll, so erhalten Sie einen Quelltext wie etwa diesen (gekürzt):

```
{ $N+ }
Unit XXX;

interface

uses WinTypes, WinProcs, WObjects, WinTools, Strings, LINKUNIT;

{Note: Additional units are linked to BUTTONS. It is necessary for
      automatic link, that they follow SOURCERS conventions.
      If these modules are created by SOURCER there is no problem!}

...

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
  TDlgWindow.Init(AParent,Name);
  StrCopy(@ResName[1],Name);Str_Pas(ResName);
  CreateColorElements;
  bu_BEISPIEL:=New(PBUTTON,InitResource(@self,id_XXX_BEISPIEL));
end;

...

procedure TXXX.wmBEISPIEL(var Msg:TMessage);
var i:word;
```

```

    d:PLINKUNIT;
begin
    vrecXXX.ExitID:=ID_XXX_BEISPIEL;
    d:=New (PLINKUNIT,init (@self, 'LINKUNIT')) ;
    Application^.ExecDialog (D) ;
    If SaveDataToRecord then exit;
end;

...

end.

```

Beachten Sie bei der Verwendung, daß entsprechend der Herangehensweise der Engine die Namenskonvention wie folgt vorausgesetzt wird:

<i>Unitname</i>	<i>Name der Unit, die den Dialog enthält</i>
<i>'T'+Unitname</i>	<i>Erbe von TDlgWindow, der den Dialog darstellt</i>
<i>'P'+Unitname</i>	<i>Zeiger auf das Dialogobjekt</i>

Anmerkung: Verwenden Sie die Borland Custom Controls, so sollten Sie keine eigenen Buttons auf die ID's wie 1,2 oder 998 legen. Dafür können die Standardmethoden OK, Cancel und Help verwendet werden (siehe Optionen/Einstellungen). Ansonsten könnte TPW Ihnen leicht doppelte Methode für gleiche ID's anmelden. Beim RC-Import blendet die Engine diese Buttons selbsttätig aus und setzt dafür die Flagge auf die obigen Methoden. Da die Help-Methode jedoch keine Standardmethode ist, wird lediglich eine Aufforderung, eine eigene Helpfunktion einzusetzen als Aktion hinterlegt.

Editfelder (alphanumerische Eingabefelder)

In ein Editfeld kann durch den Anwender Text über die Tastatur eingegeben werden, beispielsweise ein Dateiname wie im Dialogfenster DATEI ÖFFNEN. Man nennt diese Eingabefelder auch Alphanumerische Eingabefelder, da in ihnen Text geschrieben wird, der nicht durch eigene Leseprozeduren bewertet und umgesetzt werden muß wie das bei Zahlen der Fall ist. Das Auswahlfenster ALPHANUMERISCHES EINGABEFELD in der DIALOG ENGINE ist über das Icon Nummer 6 der Iconliste bzw. über die entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Das alphanumerische Eingabefeld benötigt neben der Spezifikation des ID-Wertes und des Namens lediglich einen Startwert. Hierbei handelt es sich um den zuerst nach dem Aufrollen des Dialoges erscheinenden String, der vom Nutzer als Vorschlag betrachtet werden aber auch jederzeit gelöscht und durch einen anderen ersetzt werden kann. Dieser Wert gilt ohne Ihre Eingriffe beim Programmstart und bleibt bei erneutem Aufruf des Dialogs erhalten, da der Wert im Datenrecord erhalten bleibt.

Internes Format

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 DIALOG ENGINE 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
```

Details zum generierten Quellcode

Im folgenden ein Beispiel für einen Dialog mit nur einem Editfeld (nur die betreffenden Abschnitte sind dargestellt):

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}

            id_XXX_EDITFELD=100;

TYPE       {Fuer den Dialog verwendeter Datensatz}
```

```

        TRecXXX= record
            ExitID:word;
            vEDITFELD:string;
        end;

VAR      VRecXXX:TRecXXX;

TYPE     {Definition des verwendeten Dialogfensters}
        PXXX = ^TXXX;
        TXXX = Object(TDlgWindow)
            ResName:string;
            ed_EDITFELD:PEDIT;
            {Konstruktionsmethoden}
            constructor init(AParent:PWindowsObject;Name:PChar);
            procedure SetupWindow;virtual;
            procedure wmColor(var Msg:TMessage);virtual
                wm_CtlColor;
            {Reaktionsmethoden}
            procedure wmEDITFELD(var Msg:TMessage); virtual
                id_first+id_XXX_EDITFELD;
            {Lokale Methoden}
            function SaveDataToRecord:boolean;
            procedure CreateColorElements;
            procedure DeleteColorElements;
        end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    ed_EDITFELD:=New(PEDIT,InitResource(@self,
        id_XXX_EDITFELD,255));
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            s:=vEDITFELD+#0;
            ed_EDITFELD^.SetText(@s[1]);
        end;
end;

...

procedure TXXX.wmEDITFELD(var Msg:TMessage);
begin
    DefWndProc(Msg);
end;

function TXXX.SaveDataToRecord:boolean;
begin

```

```

        SaveDataToRecord:=false;
        With VRecXXX do
            begin
                ed_EDITFELD^.GetText(@vEDITFELD[1],255);
                str_pas(vEDITFELD);
            end;
        end;

begin

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

        With VRecXXX do
            begin
                vEDITFELD:='STARTWERT';
            end;
        end.

```

Radiobuttons

Radiobuttons sind eine Gruppe von Schaltern, die Zustände beschreiben, von denen immer nur ein einziger aktiviert sein kann. Somit besteht ein Zusammenhang zwischen einer Gruppe von solchen Radiobuttons so, daß bei Aktivierung eines von ihnen automatisch alle anderen deaktiviert sind. Die Gruppe von Radiobuttons ist in der Regel mit erklärendem Text versehen.

In der DIALOG ENGINE wird das Auswahlfenster für die GRUPPE VON RADIOBUTTONS mit dem Icon Nummer 7 bzw. der unter dem Menüpunkt Elemente des Hauptmenüs zugehörigen Funktion aktiviert. Hier erscheint dann ein Fenster.

Für die Gruppe von Radiobuttons kann im Auswahlfenster ein NAME angegeben werden. Die Beschriftung jedes einzelnen Buttons ist in der jeweiligen Ressource festgelegt). Es sei angemerkt, daß die Engine keine einzelnen Radiobuttons betrachtet sondern immer nur ein kollektives Objekt, welches aus einzelnen Buttons besteht. Aus diesem Grunde wird in der Engine ein virtueller Typ TRadio definiert, welcher diese Gruppeneigenschaft umschreibt und somit als Ersatz für TGroup zu werten ist, aber auch die Radiobuttons selbst darstellt. Dieser Typ ist nur eine Konvention innerhalb der Engine, im Quelltext werden entsprechende Methoden installiert, so daß auch die Verwaltungsinstanz für die Gruppe als Quelltext vorliegt.

Weiterhin wird angegeben, welchem ID-Code der erste Radiobutton entspricht. Alle weiteren Radiobuttons erhalten dann fortlaufende ID-Codes (Annahme der Engine!). Die ANZAHL der zugehörigen Radiobuttons legt schließlich die Größe der Gruppe fest (beginnend bei 0!). Als letzte Information wird abgefragt, welcher Button BEIM START AKTIV sein soll (auch hier ist der erste Button durch eine 0 bezeichnet). Durch die hierdurch abgefragten Informationen ist die Gruppe vollständig und eindeutig beschrieben.

Internes Format

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 er eine Checkbox. Er 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 einzelnen 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 einzelnen 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
```

Details zum Quellcode

Der folgende Codeausschnitt zeigt den typischen Einbau eines TRadio-Objektes:

```

{$N+}
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_RADIOOBJEKT=104;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vRADIOOBJEKT:longint;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                vrb_RADIOOBJEKT:longint;
                rb_RADIOOBJEKT:array[0..01] of PRadioButton;

                ...

                {Reaktionsmethoden}
                procedure wmRADIOOBJEKT_0(var Msg:TMessage); virtual
                id_first+id_XXX_RADIOOBJEKT+0;
                procedure wmRADIOOBJEKT_1(var Msg:TMessage); virtual
                id_first+id_XXX_RADIOOBJEKT+1;

                ...

                procedure rb_GroupRADIOOBJEKT(n:word);
            end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    rb_RADIOOBJEKT[0]:=New(PRADIOBUTTON,InitResource(@self,
        id_XXX_RADIOOBJEKT+0));
    rb_RADIOOBJEKT[1]:=New(PRADIOBUTTON,InitResource(@self,
        id_XXX_RADIOOBJEKT+1));
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            {Radiobutton GroupSetup}
            j:=0;

```

```

        While (vRADIOOBJEKT and (longint(1) shl j)=0) and (j<=1) do inc(j);
        if j>1 then
            begin rb_GroupRADIOOBJEKT(0);vrb_RADIOOBJEKT:=1 end else
            begin rb_GroupRADIOOBJEKT(j);
                vrb_RADIOOBJEKT:=longint(1) shl j end;
        end;
    end;

...

procedure TXXX.rb_GroupRADIOOBJEKT(n:word);
    var i,c,anz:word;
    begin
        i:=0;
        repeat
            if rb_RADIOOBJEKT[i]^.GetCheck=1 then rb_RADIOOBJEKT[i]^.SetCheck(0);
            inc(i);
        until i>01;
        rb_RADIOOBJEKT[n]^.SetCheck(1);
        vrb_RADIOOBJEKT:=longint(1) shl n;
    end;

procedure TXXX.wmRADIOOBJEKT_0(var Msg:TMessage);
    var i,c,anz:word;
    begin
        rb_GroupRADIOOBJEKT(0);
    end;

procedure TXXX.wmRADIOOBJEKT_1(var Msg:TMessage);
    var i,c,anz:word;
    begin
        rb_GroupRADIOOBJEKT(1);
    end;

...

function TXXX.SaveDataToRecord:boolean;
    begin
        SaveDataToRecord:=false;
        With VRecXXX do
            begin
                vRADIOOBJEKT:=vrb_RADIOOBJEKT;
            end;
    end;

begin
    With VRecXXX do
        begin
            vRADIOOBJEKT:=longint(1) shl 1;
        end;
end.

```

Beachten Sie, daß für jeden Button ein Bit in einem Longint reserviert wird und damit prinzipiell auch eine mehrfache Auswahl als Option offenbleibt. Damit ist aber auch die Anzahl der Radiobuttons pro Gruppe auf 32 beschränkt. Dies dürfte aber keine praktische Einschränkung darstellen. Beachten Sie auch die Bemerkungen zu Autoradiobuttons im Kapitel über den RC-Import.

Checkboxes

Eine Checkbox, oder auch Markierungsfeld ist ein rechteckiger Schalter, zu dem links oder rechts vom Schalter Text gehören kann. Das Feld wird mit einem X angekreuzt, wenn der Schalter eingeschaltet ist. Die Anwendung ist dafür verantwortlich (siehe Bemerkungen zu Autocheckboxen beim RC-Import), die Kennzeichnung des Feldes zu setzen oder zurückzunehmen, wenn der Benutzer das Markierungsfeld verwendet. Das entsprechende Fenster ist unter dem Icon Nummer 8 der Iconliste bzw. unter dem Menüpunkt ELEMENT des Hauptmenüs zu finden. Hier wird im wesentlichen neben ID-Code und Name, d.h. dem im Quelltext verwendeten Bezeichner, nur noch festgelegt, wie die Voreinstellung bei Programmstart vorgenommen wird.

Internes Format

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

Die Checkbox wird hier als boolsche Variable eingeführt. Durch die ID wird die Ressource 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
```

Details zum generierten Quellcode

Hier der Codeausschnitt zu einer Checkbox:

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_TEST=100;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vTEST:boolean;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                cb_TEST:PCHECKBOX;
```

```

        ...
        {Reaktionsmethoden}
        procedure wmTEST(var Msg:TMessage); virtual
            id_first+id_XXX_TEST;
        ...
    end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    cb_TEST:=New(PCHECKBOX,InitResource(@self,id_XXX_TEST));
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            If vTEST then cb_TEST^.SetCheck(1) else cb_TEST^.SetCheck(0);
        end;
    end;
end;

...

procedure TXXX.wmTEST(var Msg:TMessage);
var b:boolean;
begin
    With VRecXXX do
        begin
            b:=cb_TEST^.GetCheck=1;
            If b then cb_TEST^.SetCheck(0) else cb_TEST^.SetCheck(1);
            vTEST:= not vTEST;
        end;
        DefWndProc(Msg);
    end;
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            vTEST:=cb_TEST^.GetCheck=1;
        end;
    end;
end;

begin
    With VRecXXX do
        begin
            vTEST:=true;
        end;
    end;
end.

```

An dieser Stelle sei auf den geänderten Transfermechanismus hingewiesen. Alle Daten werden Standardgemäß in einem Record 'vrec'+Unitname gespeichert, so daß der Dialog seinen Inhalt auch über die Existenz des Objekts hinaus behalten kann. Der allgemeine Transfermechanismus ist nicht brauchbar, da er weder Strings noch Zahlen unterstützt und damit doch recht weit von der realen Welt eines Programmiers entfernt ist. Falls Ihnen der Bedarf an statischen Daten zu hoch erscheint, verlagern Sie den Dialog in eine DLL oder in der globalen Speicher. Der praktische Vorteil des neuen Transfermechanismus ist, das Sie hier auch gleich mit den Pascaleigenen Datentypen arbeiten können, ohne sich um Konvertierungen kümmern zu müssen. Lediglich die großen Datenmengen von Listboxen werden nicht komplett gesichert. Dies dürfte aber auch nur in seltenen Fällen nötig sein ...

List- und Comboboxen

Eine Listbox ist ein Rechteck, innerhalb dem Textzeilen dargestellt werden. Der Anwender kann in dieser Liste einen oder manchmal auch mehrere Einträge markieren (die Engine unterstützt standardmäßig die einfache Auswahl). Das Listenfenster schickt dem übergeordneten Fenster eine Botschaft über den oder die markierten Einträge. Oft werden solche Listenfenster in dem Dialogfenster DATEI ÖFFNEN verwendet. Wenn die Liste der Einträge die Länge oder Breite des Rechteckes überschreiten kann, sollte das Listenfenster mit Bildlaufleisten (Rollbalken) versehen werden. Das Auswahlfenster für die Listboxen in der DIALOG ENGINE ist über das Icon Nummer 9 der Iconliste bzw. über die entsprechende Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Unter diesem Auswahlfenster LISTBOX ist neben ID-Code und NAME der Inhalt der Liste beim Start festlegbar. Hierzu sind die Funktionen HINZUFÜGEN, LÖSCHEN und ALLE LÖSCHEN gedacht. Diese bestätigen den String, der in das Eingabefeld links neben der LISTE eingeschrieben werden kann und fügen ihn hinzu oder löschen ihn je nach ihrer gedachten Funktion. Soll beim Start ein bestimmter String der Liste speziell ausgewählt und markiert sein, so ist links neben AUSWAHL für den dann in das Editfenster geschriebenen String ein Kreuz zu setzen. Auf diese Art und Weise wird für einen zu erstellenden Dialog der Inhalt einer Liste gefüllt und je nach Wunsch für ein einzelnes Listenelement eine Vorauswahl getroffen. Eine Combobox ist eine erweiterte Form der Liste. Hierbei handelt es sich um eine Liste mit einem beschreibbaren Eingabefeld. Die Combobox verbirgt sich meist hinter einem einzelnen Editfeld, das aber zu einem Recheck aufklappbar ist. Ein aktiver Eintrag der Liste wird bei seiner Aktivierung in das Editfeld kopiert. Als wesentliches Charakteristikum ist zusammenfassend zu sagen, daß eine Combobox eine aufklappbare Liste verbunden mit einem beschreibbaren Editfeld ist. Das Auswahlfenster für die Comboboxen in der DIALOG ENGINE ist über das Icon Nummer 10 der Iconliste bzw. über die entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Für das Dialogelement Combobox gelten im wesentlichen dieselben Ausführungen wie für die Listbox. Entsprechend sehen sich die hierfür zur Verfügung stehenden Auswahlfenster auch ähnlich. Bei der Combobox wird einfach nur der Inhalt der zugehörigen aktuellen Box in bekannter Art und Weise festgelegt.

Internes Format

Quelltext: <name> = TListBox /<ID> Auswahl Selektion Daten
 <name> = TComboBox /<ID> Auswahl Selektion Daten

Listboxen und Comboboxen werden in der vorliegenden Version gleichermaßen wie Listenobjekte behandelt. Beide Objekte können 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 vorgegebenen Menge zur Verfügung. Dies mag eine Einschränkung sein, bietet aber noch genügend Arbeitserleichterung. Die Option steuert das Aussehen bei Programmstart. In Auswahl wird angegeben, ob ein Element als aktives angewählt werden soll und Selektion bestimmt dann die Nummer dieses Eintrags (-1 für keinen). Als letztes folgt die Bezeichnung des Abschnitts, in dem die Listeneinträge gespeichert sind. Da ein eigener Abschnitt für den Inhalt der Listen vergeben wird, ist die Länge der Liste wohl am ehesten durch Ihr Codesegment begrenzt, wobei bei einer größeren Anzahl von Listeneinträgen zu einem speziellen Füllverfahren zu raten ist.

Beispiele:

```
Liste = TListBox /102 @0@1@1  
...
```

```
[Data section]
```

```
[1]
Eintrag 1
Eintrag 2
Eintrag 3
```

Details zum generierten Quellcode

Hier ein Beispiel für eine Listbox mit mehreren Einträgen und Vorauswahl:

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}

            id_XXX_TEST=100;

TYPE       {Fuer den Dialog verwendeter Datensatz}

            TRecXXX= record
                ExitID:word;
                vSelTEST:integer;
                vSelStrTEST:string;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
PXXX = ^TXXX;
TXXX = Object(TDlgWindow)
    ResName:string;
    lb_TEST:PLISTBOX;

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

    {Reaktionsmethoden}
    procedure WmTEST(var Msg:TMessage); virtual
    id_first+id_XXX_TEST;

    {Lokale Methoden}
    function SaveDataToRecord:boolean;
    procedure CreateColorElements;
    procedure DeleteColorElements;
end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
```

```

        lb_TEST:=New (PLISTBOX, InitResource (@self, id_XXX_TEST));
    end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            lb_TEST^.addstring('Eintrag 1');
            lb_TEST^.addstring('Eintrag 2');
            lb_TEST^.addstring('Eintrag 3');
            lb_TEST^.addstring('Eintrag 4');
            vselTEST:=2;
            lb_TEST^.SetSelIndex(vselTEST);
        end;
    end;

...
procedure TXXX.wmTEST (var Msg:TMessage);
begin
    DefWndProc (Msg);
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            vselTEST:=lb_TEST^.GetSelIndex;
        end;
    end;

begin
    With VRecXXX do
        begin
            vselTEST:=1;
        end;
    end;
end.

```

Achtung: In der jetzigen Version wird nur der Auswahlzustand gespeichert. Setzen sie zum speichern des aktuellen Textes die Funktion 'GetString' für Listboxen und 'GetWindowText' für Comboboxen ein.

Rollbalken

Ein Rollbalken, an anderer Stelle auch Bildlaufleiste genannt, wird verwendet, um einen Bereich von Werten darzustellen. Ein Rollbalken ist ein Rechteck mit Richtungspfeilen an beiden Enden. Zwischen den Pfeilen zeigt ein quadratisches Symbol den aktuellen Wert. Es können sowohl horizontale als auch vertikale Rollbalken an beliebiger Stelle des Dialogs untergebracht werden. In den Auswahlfenstern der DIALOG ENGINE ist die Auswahl der ID-Werte über die Betätigung solcher Rollbalken möglich. Das Auswahlfenster ROLLBALKEN in der DIALOG ENGINE ist über das Icon Nummer 11 der Iconliste bzw. über die entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Für den Rollbalken sind neben den üblichen Eintragungen für den zugehörigen ID-Code und dem Namen die Angabe eines Wertes für STARTWERT, MINIMUM und MAXIMUM notwendig. Der Startwert ist genau der Wert, der bei Programmstart im quadratischen Editfeld erscheint. Der Rollbalken kann dann entsprechend bis zum Minimum und entgegengesetzt bis zum Maximum durchgeschoben werden.

Internes Format

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

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

Beispiel:

Temperature = TScrollbar /107 @10 @0 @50

Details zum generierten Quelltext

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_TEST=101;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vsbminTEST:integer;
                vsbmaxTEST:integer;
                vsbpostTEST:integer;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                sbmin_TEST,sbmax_TEST,sbpos_TEST:integer;
                sb_TEST:PSCROLLBAR;
```

```

        ...
        procedure wmTEST(var Msg:TMessage); virtual
        id_first+id_XXX_TEST;
        ...
    end;

```

implementation

```

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    sb_TEST:=New(PSCROLLBAR,InitResource(@self,id_XXX_TEST));
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            sb_TEST^.SetRange(vsbminTEST,vsbmaxTEST);
            sb_TEST^.SetPosition(vsbpostTEST);
            sbmin_TEST:=vsbminTEST;
            sbmax_TEST:=vsbmaxTEST;
            sbpos_TEST:=vsbpostTEST;
        end;
    end;
end;

procedure TXXX.wmTEST(var Msg:TMessage);
begin
    DefWndProc(Msg);
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            vsbpostTEST:=sb_TEST^.GetPosition;
            vsbminTEST:=sbmin_TEST;
            vsbmaxTEST:=sbmax_TEST;
        end;
    end;
end;

begin
    With VRecXXX do
        begin
            vsbminTEST:=50 ;
            vsbmaxTEST:=0 ;
            vsbpostTEST:=100;
        end;
    end.
end.

```

TFloat und TWord - neue Datentypen für Windows

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

```
type    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:integer;
            constructor initresource (AParent:PWindowsObject;ID:Word;
                d,min,max,n:integer);
            procedure GetMinMax(var min,max:integer);
            procedure Delta(step:integer);
            function legal(var v:integer):boolean;
            procedure setval(v:integer);
            function getval(var v:integer):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. Alle anderen Funktionen sind zum internen Gebrauch der Engine bestimmt und eine Dokumentation ist im Rahmen der Quelltexte der Unit WinTools (diese stellt eine Sammlung der wichtigsten Funktionen angefangen vom Druckerkontext über Stringroutinen bis zur Systemzeit (CMOS!) und der automatischen Positionssicherung von Fenstern dar) als Option erhältlich und dienen hier nur der Funktionssicherung der Engine. Sie dürfen diese Unit für Ihre eigenen Programme verwenden, aber die Unit wie auch andere Programmteile der Engine nicht weiter veräußern!

Felder von TFloat und TWord

Value	aktueller Wert (vom letzten erfolgreichen Auslesen des Editfeldes, was bei jedem Tastebdruck erfolgt)
Default	dieser Wert wird gesetzt, wenn das Editfeld ungültige Zeichen enthält
Minimum	untere zugelassene Grenze
Maximum	höchster erlaubter Wert

Methoden von TFloat und TWord

GetMinMax	mit dieser Funktion sollten die Grenzwerte abgefragt werden
-----------	-------------------------------------------------------------

Delta	der Wert Step wird zum aktuellen Wert addiert, danach sollte das Editfeld mit SetVal aktualisiert werden
Legal	Gibt den aktuell gültigen Wert zurück und fragt das Editfeld ab. Falls dieses keine gültige Zahl darstellt, wird der Default-Wert zurückgegeben
SetVal	diese Funktion setzt den Wert und aktualisiert das Editfeld
GetVal	wie Legal, jedoch wird eventuelle Fehlerursache in einer Messagebox angezeigt

Tip: rechtsbündigen Zahlen funktionieren mit dem Stil 'mehrzeilig'

Fließkommazahlen (TFloat)

Das Auswahlfenster FLIEßKOMMAZAHLEN in der DIALOG ENGINE ist über das Icon Nummer 12 der Iconliste bzw. über die entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Es handelt sich hierbei im wesentlichen um Editfelder, die einen Wert für eine Fließkommazahl erwarten. Hierbei erfolgt automatische Wertung sowie Fehlererkennung und im Falle eines Fehlers Wertekorrektur. Neben den Eingabefeldern für den Namen und den ID-Code befindet sich das Eingabefeld für den STARTWERT, den die Fließkommazahl bei Dialogstart annehmen soll. Zusätzlich werden Angaben für das MINIMUM und das MAXIMUM, die die Fließkommazahl jeweils annehmen darf, erwartet. Werden diese Werte unter- bzw. überschritten, so erfolgt Fehlermeldung und automatische Korrektur auf den Startwert. Für den Wert der Fließkommazahl selber ist außerdem noch die Anzahl der Stellen sowohl vor dem Komma (einzutragen unter STELLEN) als auch nach dem Komma (einzutragen unter KOMMASTELLEN), wichtig.

Objektdefinition ...

Internes Format

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 Bereichsüberwachung 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 Exit-Buttons geprüft und eine Fehlermeldung ausgelöst, so daß der Dialog dann nur mit gültigen und im Bereich liegenden Zahlen verlassen werden kann!

Beispiel:

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

Details zum generierten Quellcode

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      { Im Dialog verwendete ID-Codes der Elemente }
            id_XXX_TEST=100;

TYPE       { Fuer den Dialog verwendeter Datensatz }
            TRecXXX= record
                ExitID:word;
                vTEST:extended;
```

```

        end;

VAR      VRecXXX:TRecXXX;

TYPE     {Definition des verwendeten Dialogfensters}
        PXXX = ^TXXX;
        TXXX = Object(TDlgWindow)
            ResName:string;
            ex_TEST:PFLOAT;

            ...

            {Reaktionsmethoden}
            procedure wmTEST(var Msg:TMessage); virtual
            id_first+id_XXX_TEST;

            ...
        end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    ex_TEST:=New(PFLOAT,InitResource(@self,id_XXX_TEST,
        0.0,-10.0,10.0,4,1));
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            ex_TEST^.SetVal(vTEST);
        end;
    end;
end;

procedure TXXX.wmTEST(var Msg:TMessage);
var min,max,v:extended;
begin
    ex_TEST^.GetMinMax(min,max);
    ex_TEST^.legal(v);
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            If ex_TEST^.GetVal(vTEST) then SaveDataToRecord:=true;
        end;
    end;
end;

begin

```

```
With VRecXXX do
  begin
    vTEST:= 0.0;
  end;
end.
```

Fließkommazahlen mit Rollbalken

Das Auswahlfenster FLIESSKOMMAZAHL MIT ROLLBALKEN in der DIALOG ENGINE ist über das Icon Nummer 14 der Iconliste bzw. über die entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Da es sich wieder um Fließkommazahlen handelt werden auch hier die üblichen Werte für STARTWERT, MINIMUM, MAXIMUM, STELLEN und DAVON NACH KOMMA erwartet. Da das Fließkommazahleditfeld jedoch mit einem Rollbalken versehen ist, müssen zusätzlich noch Angaben über die absolute Schrittweite bei Betätigung der kleinen Pfeile oben und unten am Rollbalken unter SCHRITTWEITE sowie die Anzahl der möglichen Schritte beim schnellen Durchscrollen (einfachem Klick auf der Rollbalkenleiste) unter SCHRITTE erfolgen.

Internes Format

```
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
```

Details zum Quellcode

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_fl_TEST=100;
            id_XXX_sc_TEST=101;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vTEST:extended;
                vstepTEST:extended;
                vsbminTEST:integer;
                vsbmaxTEST:integer;
                vsbpostTEST:integer;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                sfupdate_TEST:boolean;
                sbmin_TEST, sbmax_TEST, sbpos_TEST:integer;
                sf_stepTEST:Extended;
                sf_TEST:PFloat;
                sb_TEST:PScrollbar;

                ...

                {Reaktionsmethoden}
                procedure wmflTEST(var Msg:TMessage); virtual
                    id_first+id_XXX_fl_TEST;
                procedure wmsbTEST(var Msg:TMessage); virtual
                    id_first+id_XXX_sc_TEST;

                ...
            end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    sf_TEST:=New(PFloat,InitResource(@self,id_XXX_fl_TEST,
        10.000,0.000,100.000,7,3));
    sb_TEST:=New(PScrollbar,InitResource(@self,id_XXX_sc_TEST));
    sf_stepTEST:=1.000;
    sbmin_TEST:=0;
```

```

    sbmax_TEST:=100;
    sfupdate_TEST:=false;
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            sb_TEST^.SetRange(sbmin_TEST,sbmax_TEST);
            sf_TEST^.SetVal(vTEST);
        end;
    end;
end;

...

procedure TXXX.wmsbTEST(var Msg:TMessage);
var min,max,v:extended;
begin
    if sfupdate_TEST then exit;
    sfupdate_TEST:=true;
    sf_TEST^.GetMinMax(min,max);
    if msg.wparam<>sb_thumbtrack then
        begin
            if msg.wparam=sb_lineup then sf_TEST^.delta(sf_stepTEST) else
            if msg.wparam=sb_linedown then sf_TEST^.delta(-sf_stepTEST) else
            if msg.wparam=sb_pageup then sf_TEST^.delta(10*sf_stepTEST) else
            if msg.wparam=sb_pagedown then sf_TEST^.delta(-10*sf_stepTEST) else
            if msg.wparam=sb_thumbposition then
                sf_TEST^.setval((sbmax_TEST-sb_TEST^.getposition)
                    / (sbmax_TEST-sbmin_TEST) *
                    (max-min)+min);
            end;
            sf_TEST^.legal(v);
            sb_TEST^.setposition(sbmax_TEST-trunc((v-min)/(max-min)
                *(sbmax_TEST-sbmin_TEST)));
            sfupdate_TEST:=false;
        end;
    end;

procedure TXXX.wmflTEST(var Msg:TMessage);
var min,max,v:extended;
begin
    if sfupdate_TEST then exit;
    sfupdate_TEST:=true;
    sf_TEST^.getminmax(min,max);
    sf_TEST^.legal(v);
    if (v>=min) and (v<=max) then
        sb_TEST^.setposition(sbmax_TEST-trunc((v-min)/(max-min)
            *(sbmax_TEST-sbmin_TEST)))
        else messagebeep(0);
    sfupdate_TEST:=false;
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;

```

```
        With VRecXXX do
            begin
                vsbpostTEST:=sb_TEST^.GetPosition;
                vsbminTEST:=sbmin_TEST;
                vsbmaxTEST:=sbmax_TEST;
                If sf_TEST^.GetVal(vTEST) then SaveDataToRecord:=true;
            end;
        end;

begin
    With VRecXXX do
        begin
            vTEST:= 10.000;
        end;
    end.
end.
```

Ganze Zahlen

Das Auswahlfenster GANZE ZAHLEN in der DIALOG ENGINE ist über das Icon Nummer 13 der Iconliste bzw. über der entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Es handelt sich auch hierbei im wesentlichen um Editfelder, die einen Wert für eine ganze Zahl erwarten. Hierbei erfolgt ebenfalls, wie bei der Fließkommazahl automatische Wertung sowie Fehlererkennung und im Falle eines Fehlers Wertekorrektur. Neben den Eingabefeldern für den Namen und den ID-Code befindet sich das Eingabefeld für den STARTWERT, den die Zahl bei Dialogstart annehmen soll. Zusätzlich werden Angaben für das MINIMUM und das MAXIMUM, die die Zahl jeweils annehmen darf, erwartet. Werden diese Werte unter- bzw. überschritten, so erfolgt Fehlermeldung und automatische Korrektur auf den Startwert. Für den Wert der ganzen Zahl selber ist außerdem noch die Anzahl der Stellen (einzutragen unter STELLEN) wichtig.

Objektdefinition...

Internes Format

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!

Beispiele:

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

Details zum Quelltext

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_TEST=100;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vTEST:extended;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                ex_TEST:PWORD;

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

```

        {Reaktionsmethoden}
        procedure wmTEST(var Msg:TMessage); virtual id_first+id_XXX_TEST;

        {Lokale Methoden}
        function SaveDataToRecord:boolean;
        procedure CreateColorElements;
        procedure DeleteColorElements;
    end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
    ex_TEST:=New(PWORD,InitResource(@self,id_XXX_TEST,0,0,100,6));
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            ex_TEST^.SetVal(vTEST);
        end;
    end;
end;

...

procedure TXXX.wmTEST(var Msg:TMessage);
begin
    DefWndProc(Msg);
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            If ex_TEST^.GetVal(vTEST) then SaveDataToRecord:=true;
        end;
    end;
end;

begin
    With VRecXXX do
        begin
            vTEST:=0;
        end;
    end;
end.

```

Ganze Zahl mit Rollbalken

Das Auswahlfenster GANZE ZAHLEN MIT ROLLBALKEN in der DIALOG ENGINE ist über das Icon Nummer 15 der Iconliste bzw. über der entsprechenden Funktion unter dem Menüpunkt ELEMENTE des Hauptmenüs anwählbar.

Da es sich um Ganze Zahlen handelt, werden auch hier die üblichen Werte für STARTWERT, MINIMUM, MAXIMUM und STELLEN. Da das Editfeld jedoch mit einem Rollbalken versehen ist, müssen zusätzlich noch Angaben über die absolute Schrittweite bei Betätigung der kleinen Pfeile oben und unten am Rollbalken unter SCHRITTWEITE sowie die Anzahl der möglichen Schritte beim schnellen Durchrollen (einfachem Klick auf der Rollbalkenleiste) unter SCHRITTE erfolgen.

Internes Format

```
Quellzeile: <name> = TSCWord /ID [#hor]
           @Start@Min@Max@St@ScMin@ScMax@Inc
```

Oft ist es sinnvoll, auch Ganze Zahlen 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 TWord 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.
- 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 =TSCWord /107 @1.0 @-5 @5 @5 @1 @0 @100 @0.1
```

Details zum Quelltext

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_sw_TEST=100;
            id_XXX_sc_TEST=101;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vTEST:integer;
                vstepTEST:integer;
                vsbminTEST:integer;
                vsbmaxTEST:integer;
                vsbpostTEST:integer;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                swupdate_TEST:boolean;
                sbmin_TEST, sbmax_TEST, sbpos_TEST:integer;
                sw_stepTEST:integer;
                sw_TEST:PWord;
                sb_TEST:PScrollbar;

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

                {Reaktionsmethoden}
                procedure wmswTEST(var Msg:TMessage); virtual
                    id_first+id_XXX_sw_TEST;
                procedure wmsbTEST(var Msg:TMessage); virtual
                    id_first+id_XXX_sc_TEST;

                {Lokale Methoden}
                function SaveDataToRecord:boolean;
                procedure CreateColorElements;
                procedure DeleteColorElements;
            end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
```

```

    CreateColorElements;
    sw_TEST:=New(PWORD,InitResource(@self,id_XXX_sw_TEST,10,0,100,0));
    sb_TEST:=New(PScrollBar,InitResource(@self,id_XXX_sc_TEST));
    sw_stepTEST:=1;
    sbmin_TEST:=0;
    sbmax_TEST:=100;
    swupdate_TEST:=false;
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            sb_TEST^.SetRange(sbmin_TEST,sbmax_TEST);
            sw_TEST^.SetVal(vTEST);
        end;
end;

...

procedure TXXX.wmsbTEST(var Msg:TMessage);
var min,max,v:integer;
begin
    if swupdate_TEST then exit;
    swupdate_TEST:=true;
    sw_TEST^.GetMinMax(min,max);
    if msg.wparam<>sb_thumbtrack then
        begin
            if msg.wparam=sb_lineup then sw_TEST^.delta(sw_stepTEST) else
            if msg.wparam=sb_linedown then sw_TEST^.delta(-sw_stepTEST) else
            if msg.wparam=sb_pageup then sw_TEST^.delta(10*sw_stepTEST) else
            if msg.wparam=sb_pagedown then sw_TEST^.delta(-10*sw_stepTEST) else
            if msg.wparam=sb_thumbposition then
                sw_TEST^.setval(trunc((sbmax_TEST-sb_TEST^.getposition)
                                     / (sbmax_TEST-sbmin_TEST) *
                                     (max-min)+min));
        end;
    sw_TEST^.legal(v);
    sb_TEST^.setposition(sbmax_TEST-trunc((v-min)/(max-min)
                                           *(sbmax_TEST-sbmin_TEST)));
    swupdate_TEST:=false;
end;

procedure TXXX.wmswTEST(var Msg:TMessage);
var min,max,v:integer;
begin
    if swupdate_TEST then exit;
    swupdate_TEST:=true;
    sw_TEST^.getminmax(min,max);
    sw_TEST^.legal(v);
    if (v>=min) and (v<=max) then
        sb_TEST^.setposition(sbmax_TEST-trunc((v-min)/(max-min)
                                                *(sbmax_TEST-sbmin_TEST)))
        else messagebeep(0);
    swupdate_TEST:=false;
end;

```

```

end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            vsbpostTEST:=sb_TEST^.GetPosition;
            vsbminTEST:=sbmin_TEST;
            vsbmaxTEST:=sbmax_TEST;
            If sw_TEST^.GetVal(vTEST) then SaveDataToRecord:=true;
        end;
    end;

begin
    With VRecXXX do
        begin
            vTEST:=10;
        end;
    end.
end.

```

Statisches Element

Ein statisches Element ist ein Textfeld, das in einen Dialog eingefügt werden kann. Das statische Element ist unter dem Menüpunkt des Hauptmenüs BEARBEITEN - STATISCHES ELEMENT zu finden.

Hier sind außer ID-Code und Name des Feldes lediglich der Starttext einzugeben, der im Dialog zuerst erscheint.

Benutzen Sie statische Elemente nur dann als Objekt, wenn Sie diese als Statusanzeige oder als veränderliche Information oder Beschriftung verwenden wollen. Sonst wird unnötiger - weil nicht verwendeter - Programmcode erzeugt (gehen Sie sparsam mit den Ressourcen um).

Internes Format

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

Details zum Quellcode

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

CONST      {Im Dialog verwendete ID-Codes der Elemente}
            id_XXX_TEST=100;

TYPE       {Fuer den Dialog verwendeter Datensatz}
            TRecXXX= record
                ExitID:word;
                vTEST:string;
            end;

VAR        VRecXXX:TRecXXX;

TYPE       {Definition des verwendeten Dialogfensters}
            PXXX = ^TXXX;
            TXXX = Object(TDlgWindow)
                ResName:string;
                ts_TEST:PSTATIC;

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

                {Reaktionsmethoden}
                procedure wmTEST(var Msg:TMessage); virtual id_first+id_XXX_TEST;
                ...
            end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
```

```

        CreateColorElements;
        ts_TEST:=New(PSTATIC,InitResource(@self,id_XXX_TEST,255));
    end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            s:=vTEST+#0;
            ts_TEST^.SetText(@s[1]);
        end;
    end;

...

procedure TXXX.wmTEST(var Msg:TMessage);
begin
    DefWndProc(Msg);
end;

function TXXX.SaveDataToRecord:boolean;
begin
    SaveDataToRecord:=false;
    With VRecXXX do
        begin
            ts_TEST^.GetText(@vTEST[1],255);str_pas(vTEST);
        end;
    end;

begin
    With VRecXXX do
        begin
            vTEST:='';
        end;
    end.
end.

```

Farbsteuerung

Im Gegensatz zu Standarddialogen unter Windows ist es in DIALOG ENGINE jedem Dialog eine eigene Farbdefinition zu geben. Die Auswahl der Farbdefinitionen erfolgt über das Auswahlfenster.

Zum Verständnis der Anwendung ist zu sagen, daß diese einer Vorgehensweise entspricht, die sich im wesentlichen in die Hauptpunkte

- Elementerauswahl und
- Farbdefinition

gliedert. Zu beachten ist in diesem Zusammenhang noch, daß bei der Auswahl der Elemente die Standardelemente und die Nichtstandardelemente zu unterscheiden sind. Dieses wird nun näher erklärt:

Elementerauswahl

In der Listbox ELEMENTE sind die in einem Dialog mit Farbeinstellungen vorgesehenen Elemente enthalten. Beim Start wird eine bestimmte Standardeinstellung definiert. Diese kann während dem weiteren Arbeiten durch neue Elemente erweitert und hinsichtlich der Farbdefinition verändert oder auch beibehalten werden. Die Auswahl der Elemente erfolgt über den jeweils zugehörigen ID-Code, der im ID-Fenster angegeben ist. Will man nun ein Element zu der bereits vorhandenen und in der ELEMENT-Listbox aufgeführten Elementeliste hinzufügen, so wählt man den entsprechenden ID-Code im ID-Fenster an und fügt es durch den Endbutton NEU zu. Das gewünschte Element ist nun auch in der Listbox zu lesen. Unmittelbar hierzu gehört auch die Funktion bzw. der Endbutton LÖSCHEN. Hiermit können Elemente aus der Elementeliste auch wieder entfernt werden. Durch das Betätigen des LÖSCHEN-Buttons wird genau das Element aus der Listbox entfernt, was entweder im nichtausgeklappten Zustand in der Editzeile ausgeschrieben ist, oder vorher invers markiert wurde. Einzig das Löschen des Standards ist nicht möglich.

Standardelemente und Nichtstandardelemente

Unter den einzelnen Elementen unterscheidet man Standardelemente und Nichtstandardelemente.

- Für die Standardelemente wird jeweils die definierte Hintergrundfarbe des Dialoges, wie sie dem Standard entspricht, verwendet. Die Textfarbe kann vom Nutzer verändert werden.
- Die Nichtstandardelemente haben eine eigene Textfarbe und Hintergrundfarbe.

Die Auswahl, ob ein Element ein Standartelement ist, kann über den Checkbutton STANDARD vorgenommen werden. Bei Änderungen von Farbdefinitionen im Standardzustand wird diese gleichzeitig bezüglich aller Standardelemente vorgenommen, ansonsten bei Nichtstandardelementen nur für das jeweils durch die ELEMENTE-Listbox angewählte aktuelle Element.

Farbdefinition

Die Farbgebung erfolgt nach Auswahl des Elements bzw. für den Standardzustand, die oben erklärt worden sind, über die für die jeweiligen Farben gültigen RGB-Werte. Diese können durch den unter ROT, GRÜN, BLAU untergebrachten Rollbalken oder den Eintrag des entsprechenden Wertes in das Eingabefeld verändert werden. Die zugehörigen Hexadezimalzahlen der gesamten Farbdefinition werden darunter passiv mit angegeben. Die Festlegung ob Textfarbe oder Hintergrundfarbe erfolgt über die Radiobuttons TEXTFARBE und HINTERGRUNDFARBE. Die angewählte Farbdefinition der Elemente ist entsprechend für die aktuelle Einstellung der Radiobuttons gültig. Hierzu gehört als letztes noch das Edit-Fenster für die aktuelle Farbeinstellung des Dialogs, die sich bei Änderung der Farbeinstellung ständig mit verändert, so daß eine Vorsichtung der angewählten Dialogfarbe möglich ist.

Timer und automatische Positionierung

Timer

Die Engine beherrscht auch die automatische Generierung von Timern. Unter Windows kann man bekanntlich Systemweit bis zu 16 Timer erhalten und wird dann entweder direkt zur Bearbeitung dieses Ereignisses aufgefordert oder bekommt eine entsprechende Meldung in die Warteschlange platziert. Die Engine arbeitet dabei nach dem Meldungsprinzip. Um die einzelnen Timerereignisse unterscheiden zu können, werden sie gezählt. Als Besonderheit kann das erste Ereignis mit einer besonderen Zeitspanne versehen werden (z.B. für einen Dialog der sich nach selbst wieder vernichtet wie dem Titelbild dieses Programmes). Ebenso kann das erste Ereignis zu verschiedenen Reaktionen führen.

Das erste Ereignis kann eine Kombination von drei Reaktionen auslösen: Nach dem ersten Ereignis kann der Dialog beendet werden, ein Signal ausgegeben und die aktuellen Daten in den Datenrecord gesichert werden. Da automatisch auch alle An- und Abmeldungen erledigt werden, stellt diese Fähigkeit der Engine eine weitere Zeitersparnis dar. Beachten Sie im Beispielquelltext die Variable `TimerInstalled`. Wenn Sie den Dialog über eigene Wege verlassen, sollten Sie falls diese Variable gesetzt ist, den Timer vernichten und die Dialogobjekte mit Hilfe der Methode `DeleteColorElements` vernichten. Auch zu diesem Thema ein Codeausschnitt:

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

TYPE      {Fuer den Dialog verwendeter Datensatz}
TRecXXX= record
    ExitID:word;
end;

VAR       VRecXXX:TRecXXX;

TYPE      {Definition des verwendeten Dialogfensters}
PXXX = ^TXXX;
TXXX = Object(TDlgWindow)
    ResName:string;
    FirstTimer:Boolean;
    TimerInstalled:Boolean;

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

    end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
```

```

end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
  TDlgWindow.SetupWindow;
  With VRecXXX do
    begin
    end;
  FirstTimer:=True;
  TimerInstalled:=SetTimer(hWindow,$ff,1,nil)<>0;
  If not TimerInstalled then message('Die Systemresource für'+
    'den Starttimer ist nicht verfügbar!',1);
end;

...

procedure TXXX.wmTimer;
var i:integer;
begin
  If FirstTimer then
    begin
      If TimerInstalled then KillTimer(hWindow,$FF);
      MessageBeep(0);
      SaveDataToRecord;
      DeleteColorElements;
      vrecXXX.ExitId:=$FF;
      Enddlg(i);
    end;
  {----- Hier die Standardbehandlung für das Timerevent ergänzen! ----}
end;

function TXXX.SaveDataToRecord:boolean;
begin
  SaveDataToRecord:=false;
  With VRecXXX do
    begin
      end;
end;

begin
  With VRecXXX do
    begin
      end;
end.

```

Automatische Positionierung

Sicherlich haben Sie auch schon oft das Problem gehabt, das der Fenstermanager von Windows die Fenster genau dahin platziert, wo Sie Ihnen am wenigsten gefallen. Oder die Dialogposition ist in der Resource definiert, aber eine andere Position wäre besser. Die Option der Positionssicherung stellt sicher, daß die Dialoge (bzw. auch das Hauptprogramm wenn Sie die entsprechende Flagge ankreuzen) immer genau an der gleichen Stelle auftauchen wie beim letzten Start.

Der Mechanismus ist folgender. Die letzte Fensterposition wird über die Funktion DlgPos aus WinTools geladen. Dabei wird der Ressourcenbezeichner als Name des Dialogs verwendet. Da dem Programm eine

eigene Ini-Datei zugeordnet wird, kann der Dialog dort die Lageinformationen abspeichern und nachladen. Aus diesem Grunde wird auch die Variable ResName im Erben von TDlgWindow definiert. Da in der Methode SetupWindow schon eingültiges Fensterhandle vorliegt, kann dort die Position aus der Ini-Datei gelesen und das Fenster entsprechend umgesetzt werden. Das Pendant wird beim Beenden ausgeführt, wie es der nachfolgende Codeausschnitt zeigt:

```
{ $N+ }
Unit XXX;

interface uses WinTypes, WinProcs, WObjects, WinTools, Strings;

    id_XXX_Help=998;

TYPE      {Fuer den Dialog verwendeter Datensatz}
TRecXXX= record
    ExitID:word;
end;

VAR      VRecXXX:TRecXXX;

TYPE      {Definition des verwendeten Dialogfensters}
PXXX = ^TXXX;
TXXX = Object(TDlgWindow)
    ResName:string;
    {Konstruktionsmethoden}
    constructor Init(AParent:PWindowsObject;Name:PChar);
    procedure SetupWindow;virtual;
    procedure wmColor(var Msg:TMessage);virtual wm_CtlColor;
    procedure OK(var Msg:TMessage);virtual id_first + id_Ok;
    procedure Cancel(var Msg:TMessage);virtual id_first + id_Cancel;
    procedure Help(var Msg:TMessage);virtual id_first + id_XXX_Help;
    ...
end;

implementation

constructor TXXX.Init(AParent:PWindowsObject;Name:PChar);
begin
    TDlgWindow.Init(AParent,Name);
    StrCopy(@ResName[1],Name);Str_Pas(ResName);
    CreateColorElements;
end;

procedure TXXX.SetupWindow;
var s:string;j:word;
begin
    TDlgWindow.SetupWindow;
    With VRecXXX do
        begin
            end;
        DlgPos(hWindow,ResName,WP_Load);
    end;

...

procedure TXXX.OK;
```

```

var i:integer;
begin
  If SaveDataToRecord then exit;
  DeleteColorElements;
  DlgPos(hWindow,ResName,WP_Save);
  EndDlg(i);
end;

procedure TXXX.Cancel;
var i:integer;
begin
  DeleteColorElements;
  DlgPos(hWindow,ResName,WP_Save);
  EndDlg(i);
end;

procedure TXXX.Help;
begin
  message('Sie haben Hilfe zum Dialog TXXX angefordert. '+
    'Bitte fügen Sie einen WinHelp-Aufruf in die Methode Help'+
    ' ein, oder rufen Sie Ihre eigene Hilferoutine auf.',17);
end;

function TXXX.SaveDataToRecord:boolean;
begin
  SaveDataToRecord:=false;
  With VRecXXX do
    begin
    end;
end;

begin
  With VRecXXX do
    begin
    end;
end.

```

Autorenangaben

Ein kleiner Service für den Programmierer ist die automatische Erstellung eines Programmkopfes, in welchem die Angaben des Autors, das aktuelle Projekt, die Firma und das aktuelle Datum verzeichnet sind. Ein solcher Kopf wird in alle Dialogunits und in alle Hauptprogramme eingetragen. Die Einstellungen sind lokal zum Dialog (bzw. zur *.Dlg-Datei), werden aber bei als Grundeinstellung von der Engine geführt. Füllen Sie die Felder im Dialog Autor einmal aus, bleiben diese erhalten, bis Sie diese ändern und werden für alle neuen Dialoge genutzt.

```
{ **** }
{ * }
{ * Project : * }
{ * Unit : XXX.Pas * }
{ * }
{ * Author : Uwe Richter * }
{ * Firm : * }
{ * Adress : 1055 Berlin, Grellstraße 63 * }
{ * }
{ * Created : Mittwoch, den 29. Juli 1992 um 22:05:53 Uhr * }
{ * Update : * }
{ * }
{ * Version : Basic source created by Dialog Engine 1.5 * }
{ * }
{ **** }
```

Der obige Header zeigt den typischen Aufbau. Die letzte Zeile mit dem Verweis für die Version der Engine ist wie das Feld Update für den zukünftigen Gebrauch des Mixers gedacht, der es erlaubt bestehende Quelltexte erneut zu compilieren und die fehlenden Objekte hinzuzufügen. Löschen Sie also diese Zeilen nur, wenn Sie sicher sind daß die Engine (als Update auf den Mixer) nichts mehr mit dem Quelltext zu tun haben wird.

Programmrümpfe

Obwohl die Engine auf die Erstellung von Dialogen ausgerichtet ist, kann man mit ihr auch Hauptprogramme erzeugen. Obwohl diese Funktion in der Engine konzeptionell nur rudimentär integriert ist, bietet sie nützliche Funktionen. Das Hauptziel ist es, den erstellten Dialog schnell als Programm laufen lassen und damit testen zu können. Dazu kann man die Einstellungen im Menü Hauptprogramm/Vorgaben verwenden. Jeder Dialog bekommt ein eigenes kleines Hauptprogramm zum Test zugeteilt. Der Programmname wird als Voreinstellung aus dem Dialognamen mit vorangestelltem 'P' wie Programm erzeugt. Das Standardprogramm besteht aus einem einfachen Hauptfenster, welches als erste und einzige Aktion den Dialog startet. Falls Sie das Hauptfenster nicht sehen wollen, tragen Sie einfach TestDlg.Pas als Programmnamen ein, dann wird ein unsichtbares Hauptfenster verwendet.

Falls Sie wollen können dem Programm jedoch auch einige Standardeigenschaften gegeben werden. Zu diesen Standardeigenschaften zählen ein Menü mit Beschleunigertabelle, ein Icon und die Fähigkeit Position und Größe zu sichern/ zu laden (analog zu der Funktion bei Dialogen, aber auch mit Größensicherung). Beim Einlesen von Ressourcen werden die notwendigen Informationen eingelesen, aber als Standard nicht verwendet. Wenn Sie z.B. DEMO.RC eingelesen haben, sollten Sie ruhig einmal die Comboboxen aufklappen und sich deren Inhalt ansehen. In diesem Dialog ist ebenfalls die Steuerung der Codegenerierung untergebracht, es können wahlweise Dialog und/oder Hauptprogramm erstellt werden. Im folgenden ein Beispielprogramm:

```
uses WinTypes, WinProcs, WObjects, WinTools, ABOUT;

Type  PMainWindow = ^TMainWindow;
      TMainWindow = Object(TWindow)
          constructor Init(ATitel:PChar);
          procedure GetWindowClass(var WndClass:TWndClass);virtual;
          procedure SetupWindow;virtual;
          procedure wmclose(var msg:tmessage);virtual wm_first+wm_close;

          {***** Dies ist die Testmethode für den Dialog! *****}
          procedure Dialog(var Msg:TMessage);virtual wm_user+1;
      end;

      TEngineApp=Object(TApplication)
          procedure InitInstance;virtual;
          procedure InitMainWindow;virtual;
      end;

var EngineApp:TEngineApp;

constructor TMainWindow.Init(ATitel:PChar);
begin
    TWindow.Init(nil,ATitel);
end;

procedure TMainWindow.GetWindowClass(var WndClass:TWndClass);
begin
    TWindow.GetWindowClass(WndClass);
    WndClass.hCursor:=LoadCursor(0, idc_Arrow);
    WndClass.hbrBackground:=GetStockObject(White_Brush);
    WndClass.lpszMenuName:='MAINMENU';
    WndClass.hIcon:=LoadIcon(hInstance, 'ICON0');
end;

procedure TMainWindow.SetupWindow;
```

```

var d:PABOUT;
begin
    TWindow.SetupWindow;
    PostMessage(hWindow,wm_User+1,0,0);
    if not readwindowposition(hwindow,15,inifile,'Main','WindowState') then
        showwindow(hwindow,sw_normal);
    end;

procedure TMainWindow.wmClose;
begin
    writewindowposition(hwindow,15,inifile,'Main','WindowState');
    TWindow.wmClose(msg);
end;

procedure TMainWindow.Dialog;
var d:PABOUT;
begin
    d:=New(PABOUT,Init(@self,'ABOUT'));
    Application^.ExecDialog(D);
end;

procedure TEngineApp.InitInstance;
begin
    TApplication.InitInstance;
    hAccTable:=LoadAccelerators(hInstance,'MAINACC');
end;

procedure TEngineApp.InitMainWindow;
begin
    MainWindow:=New(PMainWindow,Init('Parent Window'));
end;

begin
    If hPrevInst<>0 then
        begin
            Message(paramstr(0)+' läuft schon!',33);
            exit;
        end;
    GetDir(0,IniFile);
    IniFile:=copy(paramstr(0),1,pos('.',paramstr(0))+'Ini';
    EngineApp.Init('EngineAPP');
    EngineApp.Run;
    EngineApp.Done;
end.

```

Einstellungen

Einige wesentliche Funktionen werden über den Dialog Optionen/Einstellungen gesteuert. Beim Erzeugen von Quelltexten ist häufig die vorherige Version vorhanden. Mit der Flagge 'Pascaldateien überschreiben' können Sie entscheiden, ob Sie schnell oder vorsichtig arbeiten wollen. Falls diese Flagge gesetzt ist, überschreibt die Engine vorhandene Dateien ohne Warnung, ist sie dagegen gesetzt, müssen Sie alles selbst entscheiden (die entsprechenden Fragen werden gestellt).

Die Standardmethoden Ok, Cancel und Help werden nur generiert, wenn beim Einlesen von Ressourcen Borland Custom Controls erkannt wurden. Falls Sie dies auch für Standarddialoge wünschen, kreuzen Sie diese Flagge an. Die Flagge selbst dient dazu, den Codeumfang bei den Dialogen zu minimieren. Die Flagge 'BCC laden' dient dazu, die Bibliothek 'BWCC.DLL', die von den Borland Custom Controls benötigt wird, durch das Hauptprogramm laden zu lassen. Demzufolge hat diese Flagge nur für das Hauptprogramm Wirkung. Die Engine setzt die Flagge, wenn beim RC-Import BCC's erkannt werden und überläßt sonst Ihnen die Entscheidung.

Die vierte Flagge betrifft den RC-Import selbst. Bei den meisten Dialogen haben statische Elemente einen hohen Anteil, aber es ist sicher nicht nötig, für alle Texte ein eigenes Objekt anzulegen. Deshalb werden zunächst keine statischen Felder eingelesen, es sei denn, sie aktivieren diese Funktion mit Hilfe dieser Flagge.

Funktionen der Unit WinTools

Die Unit WinTools stellt als wichtigste Funktion die Zahlentypen TFloat und TWord zur Verfügung und enthält darüber hinaus eine nützliche Sammlung von Routinen für alle Zwecke.

Deklaration

```
unit wintools;interface

uses windos,wintypes,winprocs,wobjects;

procedure    Alphanum           (var s:string);
function     BoldFont           (height:word):hfont;
function     Continue           (s:string;icon:byte):boolean;
function     Datestring         :string;
procedure    Delay              (ms:word);
procedure    DlgPos             (hWindow:hwnd;entry:string;save:boolean);
procedure    DrawBitmap         (dc:HDC;hbm:HBitmap;
                                xStart,yStart:integer);

function     Elapsed_days       :single;
function     Elapsed_hours      :single;
function     Elapsed_minutes    :single;
function     Elapsed_seconds    :longint;
function     Extread            (var s:string;var wert:extended):
                                boolean;

function     FileExists         (name:string):boolean;
procedure    GetBitmapSize      (hbm:HBitmap;var xRes,yRes:integer);
procedure    GetIniExtended     (Bezeichner:string;default:extended;
                                var wert:extended);

procedure    GetIniInteger      (Bezeichner:string;default:integer;
                                var wert:integer);

procedure    GetIniLongint      (Bezeichner:string;default:longint;
                                var wert:longint);

procedure    GetIniString       (Bezeichner,default:string;
                                var wert:string);

function     GetPrinterDC       :THandle;
function     HelpItem           (Item:Integer):integer;
function     Intread            (var s:string;var wert:integer)
                                :boolean;

function     LongDatestring     :string;
procedure    Message            (s:string;icon:byte);
function     NewFont            (height:word):hfont;
function     Optioncount        (s:string):word;
function     Optionstring       (s:string;no:word):string;
function     ReadWindowPosition (hWindow:HWND;Flags:Word;Filename,
                                Section,Entry:string):boolean;

function     RectMeetsDialogObject(R:TRect;hWndDialogObject,
                                hWndDialogWindow:HWND):boolean;

procedure    Restorecursor;
procedure    SetIniExtended     (Bezeichner:string;wert:extended);
procedure    SetIniInteger      (Bezeichner:string;wert:integer);
procedure    SetIniLongint      (Bezeichner:string;wert:longint);
procedure    SetIniString       (Bezeichner,wert:string);
procedure    SetWallPaper       (BmpName:String;Tiled:boolean);
procedure    SRead              (var l:string;var r:extended;
                                var fehler:boolean);
```

```

procedure      Store_reference      ;
procedure      Str_pas              (var s:string);
function       Timestring           :string;
procedure      Umlaute              (var s:string);
procedure      Up                    (var s:string);
procedure      Valid                 (var s:string);
procedure      Waitcursor;
function       WriteWindowPosition  (hWindow:HWND;Flags:Word;Filename,
                                     Section,Entry:string):boolean;

const          IniFile              :string ='Test.ini';
               HelpFile             :string ='Test.hlp';
               IniSequence          :string ='StartUp';
               wt_sound              :boolean=false;

               WP_Position           = 1;
               WP_Size               = 2;
               WP_State              = 4;
               WP_Icon               = 8;
               WP_Main               = 7;
               WP_Child              =15;

               WP_Load               :boolean=false;
               WP_Save               :boolean=true;
               WP_Restore             :boolean=false;

               ch_limit:char         =',';

```

Definition TWord und TFloat

Referenz der Funktionen und Prozeduren

Alphanum

Diese Routine verwandelt einen String in eine alphanumerische Zeichenkette.
 Gültige Zeichen: '0' .. '9', '_', 'A' .. 'Z', 'a' .. 'z'

Boldfont

Erzeugt einen fetten Font der Familie Swiss (mit genauer Darstellung).

Continue

Diese Funktion ruft eine Message auf, die eine Ja-Nein-Abfrage darstellt. Die Zahl Icon bestimmt den Titel und das Icon:

Icon and 15:	0	mb_IconStop
	1	mb_IconInformation
Icon div 16:	0	"Fehler"
	1	"Zur Beachtung"

Wenn die MessageBox Id_Yes zurückgibt, wird true zurückgegeben sonst false.

Datestring

Diese Funktion gibt das aktuelle Datum in der Form TT.MM.JJ zurück.

Delay

Da unter Windows keine einfache Verzögerung möglich ist, wird diese Funktion für unkritische Verzögerungen eingesetzt. Da sie eine Schleife ausführt, wird das System in der Zeit blockiert! Beachten Sie auch die Abhängigkeit von der Taktfrequenz des Rechners.

DlgPos

Diese Funktion dient der Sicherung und Wiederherstellung der Position eines Dialogs in die aktuelle Inidatei.

Entry bezeichnet den dafür benötigten Eintrag im Abschnitt 'Dialogs'. Benutzen sie zum Speichern das Makro WP_Save und zum restaurieren der Position WP_Load.

DrawBitmap

Die Bitmap hBM wird im Devicecontext DC ab der Position (xStart, yStart) gezeichnet.

Elapsed_days

Gibt die Anzahl der vergangenen Tage seit dem letzten Aufruf von Store_Reference als Fließkommazahl an (da die Zeit aus dem CMOS gelesen wird, ist diese unbeeinflusst von Interruptsperrungen).

Elapsed_hours

Gibt die Anzahl der vergangenen Stunden seit dem letzten Aufruf von Store_Reference als Fließkommazahl an (da die Zeit aus dem CMOS gelesen wird, ist diese unbeeinflusst von Interruptsperrungen).

Elapsed_minutes

Gibt die Anzahl der vergangenen Minuten seit dem letzten Aufruf von Store_Reference als Fließkommazahl an (da die Zeit aus dem CMOS gelesen wird, ist diese unbeeinflusst von Interruptsperrungen).

Elapsed_Seconds

Gibt die Anzahl der vergangenen Sekunden seit dem letzten Aufruf von Store_Reference als Fließkommazahl an (da die Zeit aus dem CMOS gelesen wird, ist diese unbeeinflusst von Interruptsperrungen).

ExtRead

Diese Funktion liest ein EXTENDED aus einem String. Der zugehörige Teilstring wird gelöscht und im Fehlerfall true zurückgegeben.

FileExists

Ermittelt mit Hilfe einer assign-reset-close Methode, ob eine Datei existiert.

GetBitmapSize

Gibt die Größe der Bitmap hBM in Pixeln xRes und yRes zurück.

GetIniExtended

Diese Prozedur liest einen EXTENDED aus einer Inidatei (Variable IniFile) und einem Abschnitt (Variable IniSequence) aus dem Eintrag Bezeichner. Falls der Eintrag nicht existiert, wird der Default-Wert zurückgegeben.

GetIniInteger,GetIniLongint, GetIniString

Wie GetIniExtended, nur für andere Datentypen.

GetPrinterDC

Gibt den Devicekontext des Druckers oder 0 zurück.

HelpItem

Ruft die Hilfedatei aus Variable HelpFile auf. Diese wird beim Programmstart auf den Programmnamen mit der Erweiterung HLP eingestellt. Falls Sie eine andere Namenskonvention verwenden sollten Sie die Variable HelpFile neu setzen. Falls der Wert Item 0 ist, wird der Hilfeindex aufgerufen, sonst der entsprechende Kontext-Wert (definiert im MAP-Abschnitt ihres Hilfeprojektes). Sie sollten diesen Aufruf verwenden, da Sie so alle Referenzen auf die Hilfedatei zentral steuern können.

Intread

Wie ExtRead für Integerwerte.

Longdatestring

Gibt das aktuelle Datum in der Form wie 'Dienstag, den 15. Juli 1992' zurück.

Message

Diese Funktion ruft eine Message auf, die einen OK-Button darstellt. Die Zahl Icon bestimmt den Titel

und das Icon:

Icon and 15:	0	mb_IconStop
	1	mb_IconInformation
Icon div 16:	0	"Fehler"
	1	"Zur Beachtung"
	2	"Information"

Benutzen Sie diese Funktion für Meldungen oder als WriteIn-Ersatz beim Testen von Programmen.

NewFont

Wie BoldFont, aber normale Strichstärke.

Optioncount

Zerlegt einen String in Teilstücke, die durch das Zeichen CH_LIMIT getrennt werden. Der String vor dem ersten CH_LIMIT trägt die Nummer 0.

Optionstring

Gibt den n-ten Teilstring eines Strings zurück, der durch CH_LIMIT in Teilstücke zerlegt wird.

ReadWindowPosition

Restauriert die Fensterposition des Fensters HWindow aus der Ini-Datei Filename, dem Abschnitt Section und dem Eintrag Entry. In Abhängigkeit von Flags werden folgende Teile gelesen:

```
WP_Position      = 1;  
WP_Size         = 2;  
WP_State        = 4;  
WP_Icon         = 8;  
WP_Main         = 7;  
WP_Child        =15;
```

Falls der gesuchte Eintrag nicht existiert, kommt der Fenstermanager von Windows zum Zuge.

RectMeetsDialogObject

Bei Dialogen kann man ungehindert in statischen Elementen eigene grafische Darstellungen unterbringen. Leider muß man sich selbst um das restaurieren kümmern. Falls sie wm_paint abfangen (BeginPaint .. EndPaint), so kommen Sie irgendwann an dem Problem vorbei, ob das zu neuzumalende Rechteck R das Dialogobjekt hwndDialogobjekt im Dialog hwndDialogwindow betrifft. Diese Funktion gibt gegebenenfalls true zurück.

Restorecursor

Setzt den Cursor vor dem Aufruf von Waitcursor wieder ein. Beachten Sie, daß WaitCursor und RestoreCursor paarig aufgerufen werden müssen und die Anzeige des Uhrenglases für Wartezeiten

steuern. Damit in Unterrouتين keine Probleme auftreten, können Sie einen Stapel von 9 Aufrufen berücksichtigen, Hauptsache die Aufrufe sind in der Summe paarig.

SetIniExtended, SetIniInteger, SetIniLongint, SetIniString

Setzt in der Datei Inifile (Variable) und dem Abschnitt IniSequence (Variable) den Bezeichner auf den angegebenen Wert.

SetWallPaper

Diese Funktion setzt einen neuen Hintergrund auf den Desktop. Anwendbar für Installationsprogramme.

Sread

Klein, aber nützlich wird ein Extended aus einem String gelesen, der notwendige Teilstring gelöscht und bei Fehlern die Variable Error gesetzt. Vorteilhaft ist, daß alle ungültigen Zeichen wie Buchstaben überlesen werden!

Store_reference

Diese Funktion setzt den zeitlichen Nullpunkt für die Funktionen ElapsedXX. Sie wird beim Programmstart einmal aufgerufen und holt die Referenz aus der CMOS-Uhr.

Str_Pas

Eine unkonventionelle Routine für den Einsatz von Strings als PChar. Wenn Sie einen String in der Schreibweise @s[1] an eine C-Funktion übergeben haben (z.B. SetWindowText(hWnd,@s[1])), so können sie mit dieser Funktion daraus wieder einen Pascalstring machen.

Beispiel:

```
s:='Fenstertitel'#0;  
SetWindowText(HWindow,@s[1]);  
GetWindowText(hWindow,@s[1]);  
str_pas(s);  
Message(s,17);
```

Timestring

Die aktuelle Uhrzeit wird aus dem CMOS gelesen und in der Form HH:MM:SS als String zurückgegeben.

Umlaute

Diese Funktion ersetzt in einem String die deutschen Umlaute in passende Zeichenkombinationen (z.B. 'ä' in 'ae').

Up

Wandelt einen String in Großbuchstaben.

Valid

Macht aus einem String einen erweiterten alphanumerischen.

Gültige Zeichen:

' ','@',' ','#',' ','./',' ','_','|',' ','+',' ','-','
'0'..'9','A'..'Z','a'..'z','#0',':'

WaitCursor

Setzt das Uhrenglas als aktuellen Cursor. Siehe Bemerkungen zu RestoreCursor.

WriteWindowPosition

Gegenfunktion zu ReadWindowPosition. Siehe dort.

Fontmechanismus

Die meisten Programme verwenden trotz der Vielfalt an Schriften unter Windows doch nur den Systemfont, da wohl einerseits weithin unbekannt ist, wie man die einzelnen Elemente zu neuen Schriften überreden kann oder es zuviel Aufwand kostet, den entsprechenden Overhead an Verwaltung zu implementieren. Denn es gilt ja wohl ganz besonders bei der Entwicklung von Programmen "Time is money" ...

Wer glaubt, daß ästhetische Gründe gegen das Mischen von Fonts sprechen dem seien zwei Beispiele genannt. Das erste sei die Strukturierung in Elementgruppen wie öfters durch Gruppenboxen realisiert. Diese Strukturierung ließe sich verbessern, wenn man die Attribute kursiv oder fett vergeben könnte. Leider wird dies als Elementattribut nicht unterstützt. Ein zweites Beispiel ergibt sich aus dem naturwissenschaftlichen Bereich, wo sehr oft Sonderzeichen oder griechische Buchstaben benötigt werden.

Ein weiterer Nachteil der Standarddialogboxen ist, daß als Schriften nur fette Typen zur Verfügung stehen. Eine normale Helvetica-Schrift wie in den meisten Dialogboxen des Resource-Workshop ist nicht verfügbar (das Attribut wird angeboten aber einfach ständig auf fett zurückgesetzt).

Um all diese Probleme zu beseitigen, wurde die Engine befähigt, jedem Element seinen eigenen Font zu ermöglichen. Dieser Mechanismus nutzt die Meldung WM_SetFont um einem Element seinen neuen Font mitzuteilen. Im Programm ist dies natürlich noch mit einiger Verwaltung verbunden, die jedoch die Engine erledigt. So müssen die benötigten Fonts angemeldet, erzeugt, gesetzt und anschließend wieder vernichtet werden. Wegen der Probleme mit den Ressourcen werden nur so viele Fonts erzeugt, wie notwendig. Falls Sie allen Elementen den gleichen Font zuweisen, wird nur einer erzeugt.

Die Dialogbox legt den jeweiligen Font fest. Im Gegensatz zur Standardkonvention von Malprogrammen, geht die Engine nicht auf Schriftschnitte sondern auf Schriftfamilien ein. Dies hat den Vorteil, daß nur Schriften aus dem Standardrepertoire von Windows verwendet werden und damit die Portierbarkeit gesichert ist (obwohl in Windows 3.0 die Symbolfonts seltsamerweise nicht funktionieren!) und Ihre Programme auch auf anderen Rechnern laufen. Es wäre ja sonst z.B. möglich, daß Sie den Adobe Typemanager installiert haben, die sehr schöne Schrift Eurostyle installiert haben, auf den Zielrechnern aber die Platzaufteilung ohne diesen Font gravierend anders ist.

Sie können zwar die Fontdefinition mit Hilfe der Routine MakeEngineFont durch ihre eigene ersetzen, aber aus obigen Gründen sei Ihnen zur Vorsicht angeraten. Die Engine erlaubt Ihnen unter dem Menü 'Bearbeiten' jedem einzelnen Element einen Font zuzuordnen oder zu bearbeiten, den eines Elementes allen zuzuweisen oder alle Fontdefinitionen zu entfernen.

Alphanumerische Eingabefelder

Die Engine benutzt für Editfelder einen eigenen Datentyp PWatchedit. Dieser besitzt eine Längenüberwachung und kann selektiv Zeichen zulassen oder verbieten. Das Objekt ist wie folgt deklariert:

```
const es_bell           =1;
      es_box            =2;
      es_german         =4;
      es_name           =8;

      cm_allchars       =0;
      cm_ExtAlphaNum    =1;
      cm_alphanum       =2;
      cm_User           =3;

type PWatchEdit=^TWatchEdit;
TWatchEdit=object(TEdit)
  AllowedChars:string;           {erlaubte Zeichen}
  value:string;                  {aktueller String}
  maxlength:integer;             {maximale Länge}
  warning:integer;               {aktuelle Warnstufe}
  german,                       {Umlaute erlaubt}
  bell,                         {akustisch warnen}
  box,                          {Warnbox bei zweitem Fehler}
  name:boolean;                 {erster Buchstabe groß}
  charmode:integer;             {welche Zeichen sind erlaubt}
  intern:boolean;

  constructor InitResource(AParent:PWindowsObject;ID:Word;
    max:integer;style:word;_charmode:integer);
  function GetText(var s:string):boolean;virtual;
  procedure SetText(s:string);virtual;
  procedure SetCharset(s:string);

  function Control:boolean;virtual;
end;
```

Beachten Sie, daß nur InitResource überschrieben ist, da im Moment nur die Verwendung als Dialogobject vorgesehen ist. Die Stilkonstanten es_* werden kombiniert in Style übergeben. Die Funktion GetText und SetText sind dem Stringtyp angepaßt. Wenn GetText true liefert, so mußte der Wert des Editfeldes korrigiert werden. Control wird aufgerufen um die Integrität des Textes zu prüfen und gegebenenfalls das Editfeld umzusetzen. Falls der CharMode User gesetzt ist, kann mit SetCharset die zugelassene Zeichenmenge übergeben werden.

ID

Identifikationscode im Dialogfenster, wird für den Messageverkehr verwendet

Name

Bezeichner im Programm (dieser muß der Pascalsyntax entsprechen, max. 64 Zeichen lang)

Wert nach Programmstart

Nach dem Start des Programmes wird der in diesem Feld enthaltene String in das Editfeld geladen. Dazu wird der Record 'vrec'<Unitname> entsprechend belegt und die LoadDataFromRecord-Methode mit diesem als Parameter aufgerufen. Wenn Sie eigene Startroutinen wollen überschreiben Sie die Setup-

Methode in einem eigenen Erben.
Maximale Länge: 120 Zeichen

Akustisch warnen

Diese Flagge legt fest, ob bei unzulässigen Zeichen oder Überschreitung der Länge ein akustisches Signal ausgegeben wird.

MessageBox

Wenn zweimal ein Fehler gemacht wurde, wird eine MessageBox zur Erklärung der Ursache des letzten Fehlers angezeigt.

Umlaute ersetzen

Falls deutsche Umlaute nicht erlaubt sind, werden automatisch 'ä' gegen 'ae' usw. ausgetauscht.

Namensartig

Falls diese Flagge gesetzt ist, werden die jeweilss ersten Buchstaben in Großbuchstaben verwandelt. Diese Eigenschaft sollte für Namensfelder verwendet werden.

CharMode

AlleZeichen	- alle Zeichen sind erlaubt
ExtAlphaNum	- siehe WinTools: Valid
Alphanum	- siehe WinTools: ASCII
User	- alle Zeichen als Default, sonst die Menge der Zeichen nach SetCharSet

Registrierung und Update

Diese Programmversion ist die Sharewareversion der Dialogengine 1.5. Die Version ist aus der Vollversion abgeleitet und erlaubt die dialoggesteuerte Programmerzeugung. Alle Funktionen sind ungekürzt nutzbar, nur sind einige Funktionen gesperrt.

Folgende Funktionen sind freigegeben:

- neue Zahlentypen (auch in Kombination mit Rollbalken)
- Farbunterstützung
- Borland Custom Controls
- Generierung von Hauptprogrammen mit Menüs, Accel. etc.

Registrierung hat folgende Vorteile:

- Quellcodes der Units WinTools, WinList, TDlgWnd
- Handbuch der Vollversion (gegen 10 DM Aufpreis)
- Updates und Support direkt vom Autor

Gegenüber der Vollversion sind folgende Funktionen eingeschränkt:

- ! - RC-Import (max. 10 Objekte)
- ! - Compiler für TWindow-Objekte (Ressourcen werden in Pascal-Quellcode übersetzt) fehlt
- Timermechanismus
- Fontmechanismus
- Verwendung der eigenen Custom Controls (Pas-Quellen vorhanden, Einbindung in Vorbereitung)
- max. 400 Quelltextzeilen pro Unit

Der Hilfetext bezieht sich auf die Vollversion und berücksichtigt die Einschränkungen der Sharewareversion nicht.

Beachten Sie die Idee der Shareware:

Sie haben Gelegenheit das Produkt in Ruhe zu prüfen und zahlen eine geringe Nutzungsgebühr, falls Sie mit dem Programm arbeiten wollen. Die Engine läßt Ihnen 30 Tage zur Prüfung Zeit. Danach sollten Sie die Registrierungsgebühr entrichten. Falls Sie die Idee der Engine jedoch wirklich verwenden wollen, wäre die Vollversion für Sie richtig.

Registrierung:	20,- DM
Vollversion:	138,- DM
Profiversion:	185,- DM (inklusive TWindow-Compiler und Cust.Contr.)

Bezahlung:	Firmen auf Rechnung
	Euroschek
	sonst. Zahlungsart nach Vereinb.

Hochschulrabatt	50%	(auch für Studenten gegen Kopie des Stud.ausweises)
-----------------	-----	-----------------------------------------------------

Versandgebühr:	10 DM
----------------	-------

Beachten Sie, daß sich die Vollversion auf dem direkten Weg zum kommerziellen Produkt befindet und der Preis der Vollversion sicher nicht lange zu halten ist, da sich der Funktionsumfang doch deutlich den der kommerziellen Produkte im Preisbereich von 1500 ..2700 Mark nähert. Trotzdem wird die Engine für

alle Anwender zu einem Preis verfügbar bleiben, der unter dem des TPW-Paketes liegt und damit die Preispolitik dieser Firmen (wie z.B. Microsoft mit Case:W für 2700,-DM) brechen. Wenn Sie die Idee der Engine gut finden, stützen Sie diese Absicht indem Sie sich durch eine Vollversion auch das Recht auf günstige Updates sichern (und dafür ist nahezu unendlich viel Freiraum gegeben).

Richten Sie gegebenenfalls Ihre Bestellungen oder Tips an

Uwe Richter
Grellstraße 63
O-1055 Berlin
Tel./Fax: 96 56 493

Viel Spaß mit der Engine!

Ihr Autor Uwe Richter.

Garantien

Der Autor gibt keine Garantien irgendeiner Art, weder ausdrücklich noch implizit, und keine Garantien der Verwendbarkeit oder Nichtverwendbarkeit für irgendeinen Zweck.

Ebenso wird keine Garantie für den Gebrauchswert dieser Software übernommen. Unter keinen Umständen ist der Autor für jedwede Folgeschäden, einschließlich aller entgangenen Gewinne und Vermögensverluste oder andere mittelbare oder unmittelbare Schäden, die durch den Gebrauch oder die Nichtverwendbarkeit dieser Software und ihrer begleitenden Dokumentation entstehen haftbar zu machen. Dies gilt auch dann, wenn der Autor über die Möglichkeit solcher Schäden unterrichtet war oder ist.