xBase++

Erfahrungen mit xBase 1.90

Erstellung realer 32-bit Windows mit Clipper-Basis

Für Microsoft Windows

(Version 1.0 Herbert Schaub)

Inhalt

Warum dieses Buch?

Ich setze Clipper seit vielen Jahren ein. 1992 bis 1995 hatte ich die Ehre, ein grösseres Projekt durchzuführen, bei welchem viel des bestehenden Clipper-Codes verwendet werden konnte. Der Kunde wollte aber alles in Windows. So suchte ich entsprechende Lösungen, Clipper unter Windows zum Laufen zu bringen. Damals schien das CA-Visual Objects erste Wahl. Es lief aber nie korrekt… Schliesslich stiessen wir auf Clip-4-Win von John Skelton. Damit konnten wir schnelle und stabile 16-bit Windowsprogramme basierend auf den bekannten Clipper Datenbanken und Funktionen erstellen.

Länger schon drängt sich der Wechsel auf 32 Bit Anwendungen auf. Ende 2006 begann wieder die Suche nach Lösungen mit den bekannten Vorgaben, viel der existierenden Funktionen weiter verwenden zu können Clip-4-Win wird leider nicht in die neuen Plattformen umgesetzt.
Ich fand xHarbour, welches versprach was ich suchte. Leider lief die Umgebung bei grösseren Applikationen nicht stabil genug und trotz dauernden Entwicklungen und der Open-Source-Philosophie erscheint momentan keine neue stabile Version, nur Betas.

So bin ich nun bei Xbase++ angelangt, welches nun meiner Hoffnung zum Umschreiben der bestehenden Programme viel Mut macht. Alle meine Clip-4-win-Applicationen sind in Win32api Styl geschrieben. Das heisst, dass Schlüsselwörter wie "SendDlgItemMessage" oder "CreateWindow" oder "CreateDialog" den Code prägen. Keine "@SAY" oder "GET" oder "PRINT" werden verwendet.

Um anderen Programmierern in ähnlichen Situationen weiter zu helfen, habe ich dieses Handbuch geschrieben.

Fall Sie, lieber Leser etwas nicht verstehen oder etwas vermissen, teilt dies bitte mit auf die Mailadresse Herbert@Schaub.net. Leute, die etwas unternehmen und anpacken machen Fehler - aber nur diese Leute lernen und wachsen an neuen Aufgaben!

Besten Dank
Herbert Schaub
Dezember 2008

April 2009:
Nach einem Monat verschiedener Tests, 2 Monaten Anpassung / Entwicklung und 1 Monat Verfeinerung steht unsere grösste Applikation für die Gemeinden in tollem 32-bit GUI zur Verfügung.
Primär geholfen hat das "inoffizielle deutsche XBase Forum". Dank allen dort sehr hilfsbereit und konstruktiv mitdenkenden Personen.

Installation xBase

Die Installation ist wirklich einfach. Setup starten und Anweisungen befolgen.

Etwas eigenwillig ist das Setzen des Editors für die .PRG-Dateien auf den Notepad. Deshalb empfehlen wir den erfahrenen Anwendern die Installation mit der Option "Benutzerdefiniert", um solchen nicht sehr sinnvollen Vorgaben ausweichen zu können.

Der Standardpfad ist C:\ALASKA\XPPW32.

Dort darunter sind in passenden Unterverzeichnissen die Programmteile und Beispiele abgelegt:

```
 +--BIN              Alle ausführbaren Programme (u.a. Compiler)
       +--BOOK             Hilfehandbücher
       +--LIB              Bibliotheken
       +--INCLUDE          Archiv der #include
       +--RESOURCE         Archiv der resourcen
       +--RUNTIME          Bibliotheken zur Laufzeitinstallation
       +--SOURCE           Quellendateien
            +--SYS         Systemroutinen
            +--COMPAT      Funktionen zur Kompatibilität zu Clipper '87
            +--SAMPLES     Beispielprogramme
```

Dazu werden die Umgebungsparameter erweitert oder müssen angepasst werden auf:

```
PATH =            C:\ALASKA\XPPW32\BIN;C:\ALASKA\XPPW32\LIB;
 LIB =              C:\ALASKA\XPPW32\LIB;
 INCLUDE =         C:\ALASKA\XPPW32\INCLUDE;
 XPPRESOURCE =   C:\ALASKA\XPPW32\RESOURCE;
```

Folgende Dateierweiterungen sind wichtig:

.CH Include-Dateien mit entsprechenden Direktiven. Während des Compilierens werden diese Dateien eingebunden.
.PRG Wer kennt diese nicht? Die Clipper-Source-Dateien.
.XFF Vom Formular-Designer erstellte Dateien.
.XPJ Dies sind die Projektdateien, die der ProjectBuilder verwendet. Ein Doppelklick startet die Erstellung der in .XPJ angegebenen Programme.

Mittlerweile sind auf der Alaska-Homepage einige Hotfixes zu finden. Diese gemäss Anleitung nachträglich installieren. Dies sind in der Regel .LIB-Dateien, welche in den entsprechenden Unterverzeichnissen ersetzt werden müssen.

Bevor Sie beginnen

Objektorientiert oder nicht?

Das klassische Clipper erlaubte keine objektorientierte Arbeit. Ja, es gibt Ansätze in z.B. das Objekt TBrowse(). Aber generell arbeitet muss man eben klassisch d.h. ablauforientiert arbeiten, die Programmierer geben vor, was wie zu geschehen hat.
Ein transformieren solchen bestehenden Codes zu echtem Windows-Code (GUI) ist nicht möglich.
Demgegenüber bietet Xbase sehr gute Tools zum Ueberführen in Pseudo - Dialoge, die aus den @SAY und @GET erfolgen. Die Migration kann also langsam erfolgen, hindert einen aber nicht, bei 100% echtem GUI alles (ausser nicht betroffene Funktionen) neu zu schreiben.

Uebersichtlicher wird’s nur bedingt, da pro Fenster eigene Abläufe definiert werden müssen, die auf Klassen und Methoden aufbauen. Gutes Design ist deshalb wichtig!

Also, die einzig wahre Lösung zum effizienten Windows Programm unter Xbase (und auch anderen GUI-Compilern) ist die Verwendung von Klassen und Methoden.
Da aber die mitgebrachten Hilfsmittel bereits einiges "von selbst" erstellen, muss man nicht bis in die weitesten Tiefen davon Wissen besitzen.

Windows Vista / Windows 7

Sie müssen alle .EXE im Unterordner BIN mit "als Administrator ausführen" geflaggt werden. Andernfalls erhalten Sie die Fehlermeldung "Error in saving Message".

Die Hilfedateien - die .HLP im Unterordner BOOK - laufen nicht mehr unter Vista. Man muss von der Microsoft-Webseite die KB917607 herunterladen und installieren.

Threads

Jeder Thread führt einen Event-Loop, fragt also unabhängig voneinander die kommenden Ereignisse ab. Einfache Programme führen in der Hauptprozedur einen einzigen sog. Thread. In der 32-bit Umgebung sind nun mehrere solcher Threads möglich. Der Einsatz mehrerer Threads soll aber überlegt sein.

Was spricht wann dafür?
- Es ist der einfachste Weg, um Alias und Pointer-fehler zu vermeiden, wenn Eingabefenster mit den gleichen Datenbanken laufen müssen. Jeder Thread hat seinen eigenen Speicherbereich und entsprechend geschützte Arbeitsbereiche.
- Wie bereits erwähnt, hat jeder Thread einen eigenen Event-Loop und entsprechend seine eigene Event-Warteschlange.

Was spricht dagegen?

- Bei mehreren Threads sind diese parallel aktiv. Die Kontrolle der Events muss genau kontrolliert werden.

Ein typisches Beispiel ist eine Applikation mit mehreren Mandanten (jeder Mandant hat dieselben Datenstrukturen), welche parallel - je in einem eigenen Thread - geöffnet werden können.

Erster Start

Beginnen Sie mit dem Dokument " (Alaska, 2006) Ihr Einstieg in 3 nur Stunden" (Gstart_d.pdf). Die Datei befindet sich im Stammverzeichnis der Installation (Default = C:\ALASKA\XPPW32).

Auch nicht fehlen darf das Inoffizielle XBase Forum (Inoffizielles deutsches Xbase-Forum) www.xbaseforum.de. Suchen Sie sich durch und Sie werden sehr viele Hinweise und interessante Dialoge finden. An dieser Stelle schon mal danke an alle dort schreibenden Personen.

Oeffnen Sie auch die Hilfedatei (ALASKA Software)XPPRef.HLP.

Arbeiten Sie sich durch die Beispiele!

"Hello World"

Unter Windows Vista ist es wichtig, dass wie bereits erwähnt alle .EXE (im Unterverzeichnis BIN) mit der rechten Maustaste mit der Option "Als Administrator ausführen" setzen. Andernfalls erhalten Sie eine Fehlermeldung.
Dieses erste Programm wird schnell erstellt sein und laufen.

Auf die weiteren DOS-Orientierten Dinge ("altes original Clipper") gehe ich weiter nicht ein, da ein Umwandeln derart alten Codes wohl kaum rentiert und ein Neuschreiben ohnehin Sinn machen wird.

Erstes GUI-Programm

Ab Seite 10 wird das Programm AppSys erklärt.

```
#INCLUDE "AppEdit.ch"

PROCEDURE Main
Use adressen
APPEDIT;
  POSITION LEFT,CENTER;
  STYLE FANCY
APPDISPLAY

RETURN
```

Dieses Programm sollte funktionieren - vergessen Sie beim Linken die Option /PM:PM nicht.

Fehlermeldungen

Bereits die erste einfache Erweiterung provoziert einen Compilerfehler. Möglicherweise liegt dies an der Demoversion 1.90.

```
#INCLUDE "AppEdit.ch"

PROCEDURE Main
Use adressen
APPEDIT;
  POSITION LEFT,CENTER;
  STYLE FANCY
APPDISPLAY MODELESS

APPBROWSE;    // bringt einen Fehler ???
  POSITION RIGHT,CENTER    // mit oder ohne diese Linie!

APPDISPLAY
RETURN
```

```
D:\prg\xbase>xpp appedit
Xbase++ (R) Compiler 1.90.331 Apr 26 2006
Copyright (c) Alaska Software. All rights reserved.
File: appedit.prg Line:   17
appedit.prg(14:0): error XBT0200: Syntax Error
1 error(s) found in file appedit.prg!
```

Linie 17 ist die Linie zwischen dem "Return" und dem "Appdisplay". Nicht gerade hilfreich, auch was die Art des Syntax-Errors betrifft.

Der Debugger

Mitgeliefert ist ein Debugger zum - von Clipper her bekannten Tracen der Programme. Aufruf XPPDBG. Das .exe steht im Unterordner BIN.

DBU

Wirklich keine schlechte Idee, ein neues DBU zu kreieren. Alles bleibt (fast) beim Alten, aber die Applikation ist eben modern punkto Programmcode - blockiert vor allem nicht mehr Systemressourcen wegen des Inkey()-Befehls.

Denken sie daran, dass das Inkey() zu Inkey(0.1) geändert werden muss. Die Aenderungen sind im Anhang erklärt.

Das DIR *.PRG > DBU.LST erzeugt nicht die korrekte Liste, Sie müssen alle überflüssigen Informationen wegnehmen, so dass nur die Dateinamen stehen ohne Datumsangabe und auch keine Datenträgerinformationen.

```
DBU.PRG
DBUCOPY.PRG
```

```
DBUEDIT.PRG
DBUHELP.PRG
DBUINDX.PRG
DBUNET.PRG
DBUSTRU.PRG
DBUUTIL.PRG
DBUVIEW.PRG
```

Das DIR *.PRG > DBU.LST erzeugt nicht die korrekte Liste, Sie müssen alle Zusatzinformationen (Datenträgerinfo usw.) weglöschen.

Allerdings bleiben noch Fehler, die ohne Nachbearbeitung den "neuen" DBU nicht einsatzbereit machen: z.B. das Aendern einer Struktur bringt schnell Fehlermeldungen. Aber es geht ja um den Effekt des Compilierens.
Eine fertig erstellte, funktionstüchtige Version des DBU kann beim Autor angefordert werden.

Der Form Designer

Der Form Designer ist eines der Kernstücke, denn ohne Formulare läuft kein Windows-Programm. Grundsätzlich wird hier die Struktur der Dialoge und deren Aufbau definiert. Die detaillierte Logik folgt später.
Im Unterverzeichnis BIN steht die Applikation XppFD.EXE.

Generell ist der Form Designer nicht überzeugend. Viele Controls fehlen und die Darstellung zeigt gröbere Mängel (z.B. das Fenster mit den Einstellungen mit z.T. unleserlichem Text)..
Um aber einen ersten Eindruck des entstandenen Codes zu finden, ist er ganz gut. Es ist generell besser, den Class-Code zu erstellen.
Achtung: Ein neuerliches Erstellenlassen des Codes überschreibt das Bisherige - eventuell von Hand erfolgte Aenderungen gehen verloren!

Externe Bibliotheken

3rd Party Programme oder eigene Bibliotheken können auf verschiedene Arten eingebunden werden

- Innerhalb der .PRG-Datei
- Innerhalb des Projektes

Im Programm wird mit der Direktive #pragma gearbeitet.

```
*pragma LIBRARY("mylib1.lib", "mylib2.lib")
```

Oder die Bibliotheken werden im PROJECT.PRJ angegeben.

```
LIB  mylib1.lib
```

Kompilieren und Linken

Der Compiler ist das XPP, welches im Ordner BIN der Alaska-Installation zu finden ist.

```
D:\prg\xbase>xpp appedit
```

Eventuelle Bitmaps werden in einer .ARC-Datei gespeichert und umgewandelt.

```
D:\prg\xbase>arc test.arc
```

Ergebnis ist eine .RES-Datei.

Gelinkt wird so:

```
D:\prg\xbase>alink test.obj tes.res /PM:PM
```

Das PM:PM ist notwendig, damit der Linker weiss, dass eine GUI-Applikation entstehen soll.

Testen:
Compilieren mit

```
D:\prg\xbase>xpp appedit /TESTING
```

Im Code mit

```
#IFDEF TESTING
…
#ENDIF
```

Den Testbereich codieren.

Der Projekt Builder

Meistens hat man ja mehr als nur eine Datei mit der Source dabei, sondern Mehrere bis Viele. Diese zum Projekt gehörenden Dateien werden zusammengefasst und im Project Builder erstellt.
Das Pbuild.exe ist das Programm dazu. Als Ergebnis entseht eine Projektdatei mit der Endung .PRJ. Default ist PROJECT.PRJ. In diesem Falle reicht ein einfacher Aufruf:

```
D:\prg\xbase>pbuild
```

Der erstmalige Aufruf generiert das Projekt. Optionen dazu sind in der Dokumentation von Alaska zu finden. Dazu wird eine Datei mit der Endung .LST generiert, die eine Liste aller beteiligten Dateien enthält

```
D:\prg\xbase>pbuild @test.lst
```

Anschliessend wird die entstandene Datei Project.xpj entstanden sein. Diese wird ein erstes und einziges Mal mit der Option -g aufgerufen, damit eventuell Fehlende Parameter eingebaut werden.

```
D:\prg\xbase>pbuild -g
```

Einsatz von xBase++

Gliederung

Wie bereits erwähnt, interessiert in diesem Buch nur der Ansatz mit objektorientertem Vorgehen, ohne auf Clipper-Code aus DOS-Zeiten Rücksicht zu nehmen.

Wie soll nun eine Applikation im OO-Modus aussehen?

Notwendige Teile:

- Mainprozedur mit dem Eventloop

enthält die Standardprogramme AppSys, DBESys, AppQuit

- ein Prg mit einigen Funktionen

ev. In einer DLL

- Dateiklasse

öffnen, schliessen, suchen schreiben etc.

- das Hauptfenster mit dem Menu

(Menu ev. separates Prg)
Toolbar
Tooltip
Statusbar

- für jedes Fenster (Dialog) ein eigenes Prg
- Hilfesystem

Wichtige Fragen:

- Wie erstelle ich Datenbanken und dazugehörende Indexe?
- Wie fülle und bearbeite ich die Daten in den Datenbanken?
- Wie frage ich die Daten ab und wie werden diese dargestellt?
- Wie sicher müssen die Daten vor Zugriffen geschützt werden?

Diese grundlegenden Punkte beeinflussen das Design und die Idee der Applikation. Dies muss geplant werden. Aber Achtung: Event driven heisst, dass der Anwender später irgendwo etwas auslösen kann. Deshalb sind Abläufe anders als in "alten" DOS-Programmen.

Elemente eines Xbase+ Programms

Bestimmte Regeln definieren die Struktur der Programme

- Procedure Appsys()
  - Läuft vor die Hauptprozedur ab. Hier können globale Werte und Parameter gesetzt werden.
  - Achtung: Appsys läuft vor dem Initialisieren der DBEsys.prg (Initialisieren der Datenbank-Engine). Deshalb dürfen hier keine Datenbanken geöffnet werden.
  - Achtung: Fehlt die Prozedur Appsys(), generiert der Linker selber eins. Das Ergebnis ist ein störendes CMD-Fenster!

- Procedure Main
  - Enthält als Hauptsache den Event-Loop und die Erstellung eines Haupdialogfensters. Parent ist das AppDesktop().
  - Sollte den Aufruf SetAppWindow(oDlg) enthalten, dies definiert den hier verwendeten Dialog als Standard-Applikationsfenster. Dieses Fenster kann später durch Aufruf der Funktion überall in einem Programm abgefragt werden, ohne dass es explizit in einer Variable referenziert sein muss.
- Dialoge
  - Pro Dialog gibt's im Prinzip eine Klasse, die der Uebersicht halber in einer eigenen .PRG lagert.
- Allgemeine Funktionen und Prozeduren
  - Laufen in separaten (für grössere Dinge) oder in Containern zusammengefasst in einer .PRG. Aufrufe dazu nach Bedarf.

Menü

Erstellen:

```
   CreateMenuSystem( SetAppWindow():menuBar() )
```

Der Code dazu:

```
PROCEDURE CreateMenuSystem( oMenubar )
LOCAL oMenu, oSubMenu
   oMenu       := XbpMenu():new( oMenuBar )
   oMenu:title := "~Daten"
   oMenu:create()
// jetzt kommt das Handeln!
   oMenu:itemSelected := {|nItem| MenuDispatcher( 1000+nItem ) }
   oMenu:addItem( {"~Login"  , NIL} )
   oMenu:addItem( {"~™ffnen" , NIL} )
   oMenu:addItem( {NIL, NIL  , XBPMENUBAR_MIS_SEPARATOR, 0} )
   oMenu:addItem( {"~Ende"   , NIL} )

   oMenubar:addItem( {oMenu, NIL} )
RETURN

PROCEDURE MenuDispatcher( nSelection )
   DO CASE
   CASE nSelection == 1001
//      QOut( "Gew„hlt ist: Datei Neu"  )
   CASE nSelection == 1002
//      QOut( "Gew„hlt ist: Datei ™ffnen" )
   CASE nSelection == 1004
//      DoEnde(.F.)
     QUIT
   ENDCASE
RETURN
```

So entsteht ein Menüpunkt mit 3 Unterpunkten.

Was passiert beim Mausklick auf ein Menü?
Die Prozedur Menudispatcher wird durchlaufen.

Man kann aber auch dem AddItem direkt einen Codeblock mitgeben, so wird dieser durchlaufen.

```
oMenu:addItem( {"~Beenden", {|| AppQuit("MeinProgramm",.T.) } } )
```

Der Codeblock zum ausführen wird an 2. Stelle angegeben. Der 1. Parameter ist numerische Position, der 3. Parameter das Menüobjekt.

Es können auch Submenüs eingefügt werden

```
oSubMenu       := XbpMenu():new( oMenu )
oSubMenu:title := "~Hier ist ein Untermen• "
oSubMenu:create()
oSubMenu:addItem( { "Erster"     , {|| QOut( "Erster Eintrag"  ) } } )
…
```

Dieses wird dann dem Menü an entsprechender stelle eingefügt:

```
oMenu:addItem( { oSubMenu, NIL } )
```

CheckMenü
Im Codeblock wird an erster Stelle das Flag gesetzt:

```
oMenu:addItem( { "~Untermenü", ;
           {|nItem,lCheck,obj| ;
              lCheck := obj:isItemChecked(1), ;
              obj:checkItem( 1, !lCheck )   , ;
              IIf( lCheck, obj:enableItem(2), obj:disableItem(2) ) } } )
```

Zugleich wird hier noch gezeigt, wie ein Menüpunkt disabled oder enabled werden kann.

Browse

Xbase kennt innerhalb der GUI-Funktionalität zwei vorprogrammierte Browse-Elemente:
- Quickbrose()
- XbpBrowse()

Das Quickbrose hat weniger Einstellmöglichkeiten, kann aber bereits angenehm eine Datenbank oder einen Array in Tabellenform anzeigen.

Aufbau:
- Kopfzeile
- Datenbereich
- Fusszeile
- Scrollbalken (horiz. und/oder vert.)
- Weitere Elemente wie Bitmaps (z.B. für Knöpfe) müssen als Child-Elemente hinzugefügt werden.

Die Kopf- resp. Fusszeile werden mit einem Präsentations-Array dargestellt.

Aufbau:
```
{{ XBP_Konstante die geändert werden soll, XBP_Konstante für den Inhalt}...}
```

Diese Parameter können bei Laufzeit verändert werden:
```
::oMLE1:setPresParam(XBP_PP_BGCLR,GRA_CLR_YELLOW)
```

Erstellung:
Das Browse-Element gehört in ein Mutter-Fenster (XbpDialog). Das XbpBrowse wird als Kind-Fenster definiert. Die Grösse wird dem Mutterfenster angepasst, falls erwünscht.

Ein Beispiel dazu steht unter
```
C:\ALASKA\XPPW32\source\samples\basics\guibrow
```

Fehlerbehandlung

Um nicht das Programm zu einem Absturz (die berühmte Fehlermeldung mit dem Trace-back und den unumgänglichen "QUIT") führen, kann dies mit einem Errorblock abgefangen werden.

```
      bOldError := ErrorBlock( {|e| Break(e)} )
```

Im Falle eines Fehlers wird an der Stelle des Errorblocks gehandelt:

```
   BEGIN SEQUENCE
//   oControl:create(,, {10,60},{610,370} )
   RECOVER USING oError
      MsgBox( "Fehler bei…..   …" )
      QUIT    // oder auch ein RETURN - heisst: diese Prozedur abbrechen und ins
Programm zurückkehren
   END SEQUENCE
   ErrorBlock( bOldError )
```

Threads

Ein neuer Thread wird immer im Parent-Fenster gestartet. Nie im Childfenster oder in Routinen. Am besten verwaltet eine kleine Funktion den Start des neuen Threads. So kann z.B ein Menü oder auch ein Pushbutton diese Funktion starten.

```
      FUNCTION LaunchThread(cThreadname,cArg)
LOCAL oThread
      oThread: := Thread():new()
oThread:start(cThreadname,cArg)
RETURN oThread
```

cThreadname ist der Name der Funktion oder der Prozedur, die innerhalb des neuen threads gestartet wird. Das cArg ist ein Argument, welches der Funktion cThreadname mitgeliefert werden kann.
Die Funktion mit dem Namen der Variablen cThreadName muss nun einen eigenen Event-Loop abfangen und auch eigene Workspaces laden und verwalten. Dieser Event-Loop ist der erste Event-Handler dieser Funktion.
Wichtig ist auch der Abschluss des Threads, wenn die Funktion beendet wird:

```
      oThread():quit()
```

Abfragen, in welchem Thread man gerade arbeitet, kann man.

```
      oThread := ThreadObject()
```

Abfrage, ob ein Thread noch "lebt":

```
      lalive := oThread:aktive
```

mit den Methoden :atStart resp. :atEnd kann Zusätzliches zugewiesen werden.

Variablen in Threads
Die Sichtbarkeit von Variablen ist nicht mehr so einfach. Generell gilt:

- LOCAL und PRIVATE sind nur innerhalb eines Threads sichtbar
- PUBLIC und STATIC sind in allen Threads sichtbar.

Vorsicht also, wenn Publics oder Statics verändert werden (was ja generell auch gilt)!

Denken Sie daran, dass in jedem Thread auch die Grundeinstellungen (z.B. SET DATE) wieder erfolgen müssen.

Klassen

Objektorientiert Arbeiten heisst zwingend mit Klassen arbeiten. Zudem hilft die Arbeitsumgebung von XBase mit vielen eigenen Klassen.

Variablen

Innerhalb einer Methode benutzte Variablen sollten möglichst LOCAL sein. Andernfalls ist der Typ PROTECTED sinnvoll.
EXPORTED wird dann nötig, wenn z.B. aus einem Code-Block oder von aussen auf eine iVar zugegriffen wird. Allerdings soll man eigentlich keine EXPORTED iVars verwenden, sondern nur PROTECTED und dann mit Access / Assign / oder normalen Methoden den Zugriff regeln (Kapselung).
HIDDEN: Methoden/Vars sind nur innerhalb der Klasse und Instanzen sichtbar.
PROTECTED: Methoden/Vars sind nur innerhalb der Klasse sichtbar.
In beiden Fällen kann nicht von aussen drauf zugegriffen werden!

Variablen definieren

In der Klasse

```
CLASS AAA FROM XbpBrowse
   EXPORTED:
      METHOD init()
      VAR nProzent
ENDCLASS
```

Definition im Init-Teil

```
METHOD AAA:init()
   ::nProzent := 0
RETURN self
```

Variablen übergeben

Die Abfragen dazu:

```
   aSize := SetAppWindow():currentSize()
   aPos  := SetAppWindow():currentPos()
```

Variablen ansprechen

`::MyVar`	Nur innerhalb einer Methode möglich, da SELF.
`MyClass:MyVar`	Von aussen

Drucken

Das Drucken ist nicht sehr übersichtlich, da sehr viele Optionen offenstehen. Hauptelement ist das Xbase-Part XbpPrinter.

Presentation-Space

Um Drucken zu können, braucht es einen sog. Presentation-Space. Dieser wird letztendlich zur Ausgabe auf den Drucker verwendet. Erst wenn alle Information darauf ist, wird losgesendet. Das heisst, es spielt absolut keine Rolle wann wo auf einem Blatt was erzeugt wird.

```
FUNCTION PrinterPS( cPrinterObjectName ,aSize)
LOCAL oPS  // ,oPrinter
LOCAL oDC := XbpPrinter():New()  // Default - Printer nehmen
//     oDC:enablePrintToFile := .f.
//     oDC:printToFile := .f.
//    nStatus := oDC:PrinterStatus()
 oDC:Create( cPrinterObjectName )

// Konfiguriertes Drucker Objekt erzeugen
oPrinter := oDC:display()    // entspricht oPS:device()
//oDC:destroy()

IF oPrinter <> NIL
//  oPrinter:setOrientation(XBPPRN_ORIENT_LANDSCAPE)
  oPS   := XbpPresSpace():New()
       oPrinter:setOrientation(XBPPRN_ORIENT_LANDSCAPE)
  aSize := oDC:paperSize()
             // Größe des bedruckbaren Bereichs
  aSize := { aSize[5] - aSize[3], aSize[6] - aSize[4]  }
  oPS:Create( oDC, aSize, GRA_PU_LOMETRIC )   // auf mm einstellen
ELSE
  oPS := NIL
ENDIF

RETURN oPS
```

Die Masseinheit des Druckers kann beeinflusst werden:

```
  GRA_PU_ARBITRARY  Beliebige Einheit. Das Koordinatensystem
                    wird auf den Viewport skaliert.
  GRA_PU_PIXEL *)   Eine Einheit ist ein Pixel
  GRA_PU_LOMETRIC   Eine Einheit ist 0.1 Millimeter
  GRA_PU_HIMETRIC   Eine Einheit ist 0.01 Millimeter
  GRA_PU_LOENGLISH  Eine Einheit ist 0.01 Inch
  GRA_PU_HIENGLISH  Eine Einheit ist 0.001 Inch
  GRA_PU_TWIPS      Eine Einheit ist 1/1440 Inch
*) Standardwert
```

Die Zeilenhöhe bei Text:

```
 nFontHeight := oPS:setFont():height
```

Fenster

Fenster werden unter GUI mit Xbase-Parts erstellt:
- XbpDialog
- XbpBrowse
- XbpFileDialog
- XbpFontDialog
- XbpHelp
- XbpHTMLViewer

Das Dialog-Element ist natürlich das entscheidende Element.
Das Fenster des hautptprogramms ist das Mutterfeld. Die darin Aufgerufenen sind Kinder (Child-Window) davon.
Es gibt zwei Typen:
- MDI Client
  - Ist innerhalb des Mutterfensters gefangen. Wird zusammen mit dem Mutterelement verkleinert und auch mit verschoben.
- SDI-Client
  - Eigenständiges Fenster mit AppDesktop() als Parent.

Diese Fenster können in der Windows-Taskliste erscheinen.

Beziehungen innerhalb der Elemente in einem Fenster
Alle Beziehungen werden in der Klasse XbpPartHandler verwaltet. Sobald ein Xbp die Methode create() ausführt, wird sequentiell ein internes Array aufgefüllt. Die Child-Liste entsteht. Zugegriffen wird mit der Funktion Childlist():

```
oButton  := xXbp:ChildList()[2]
```

Das 2. Element hier ist ein Button.
Die Reihenfolge der erstellten Parts definiert auch deren gegenseitige Überlagerung. Das Jüngere überdeckt das Ältere.

Elemente

Position / Grösse

Die Abfragen dazu:

```
aSize := SetAppWindow():currentSize()
aPos  := SetAppWindow():currentPos()
```

Enabling/Disabling

The element, which needs control (blinking cursor or highlighted) is active as follows:

```
::Element:Enable := .T.
// Enable
::Element:Enable := .F.
// Disable
```

Child Fenster

- Ein Child kann Sinn machen. In der Regel ist es sinnvoll, das Childfenster beim Schliessen des Parents auch zu schliessen.

Das Parent erfragen:

```
oParent := GetParentForm(oChild)
```

Die Kinder abfragen:

```
//->oDlg sei der Parent aller Listboxen des Dialoges

oDlg:oListBox:itemMarked := {| uNIL1, uNIL2, o | SetOtherListBoxes( oDlg) }

Func SetOtherListboxes( oDlg )
Local aList := oDlg:ChildList()
Local i

for i:=1 to len( aList)
  if Valtype(aList[i]) == "O" .and. aList[i]:IsDerivedFrom() == "xbpComboBox"
     if aList[i]:hasInputFocus() == .F.     // betrifft andere Listbox
       ->hier dann die gewünschte Aktion
     endif
  endif
next
Return NIL
```

Man kann auch über ein Child, diesen Parent erfragen, um so wieder zu allen Kindern zu gelangen::

```
oParent := oXbp:setParent()
aList:=oParent:childList()
```

Fokus setzen

SetpaaFocus setzt einem XbasePart den Focus.

```
oFocus := SetappFocus(oParent)
```

oParent ist das Element, welches den Focus erhalten soll (z.B. oDlg:picklist:leftlistbox)

Fokus abfragen

```
SetappFocus()
```

Ohne Argument.

Darstellung

Die Darstellung erfolgt mit dem sog. Präsentationsparameter. Diese sind pro XbpPart (abgeleitet von XbplWindow) definiert. Der Parameter wird durch ein zweispaltiges Array dargestellt:

```
  aPP := { { XBP_PP_FGCLR       , GRA_CLR_RED         }, ;
```

```
          { XBP_PP_COMPOUNDNAME, FONT_DEFFIXED_SMALL }  }

   oXbp := XbpSle():new( ,, { 100, 150 }, { 160, 25 }, aPP )
   oXbp:create()
```

Sobald etwas anders dargestellt werden soll als Windows vorgibt, kann dadurch angepasst werden.
Zu Lebzeiten eines Elementes wird die Aenderung so vorgenommen:

```
Obj:setPresParam(aPP)
```

Alle xBase Parts besitzen folgende Parameter:

```
  XBP_PP_BGCLR          Background color     :setColorBG()
  XBP_PP_FGCLR          Foreground color     :setColorFG()
  XBP_PP_COMPOUNDNAME   Font name as string  :setFontCompoundName()
  XBP_PP_FONT           XbpFont object       :setFont()
```

Tabs

Mit Tabs können zusammengehörende Themen in einem Dialog.aufgeteilt dargestellt werden Wer kennt sie nicht. Praktisch und willkommen.
Umsetzung mit dem XbpPart XbpTabPage(). Wichtig sind die beiden Positionen des Tabs, die von Element zu Element sich verändern, sonst sind diese unsichtbar.

```
   ::TabPage2:= XbpTabPage():new( ::drawingArea, , {12,12}, {568,396}, { {
XBP_PP_BGCLR, XBPSYSCLR_BUTTONMIDDLE } } )
   ::TabPage2:caption := "~Familie"
   ::TabPage2:minimized := .F.
   ::TabPage2:tabStop := .T.
   ::TabPage2:preOffset := 15
   ::TabPage2:postOffset := 70
```

In den Dialogdefinitionen müssen bei alle in einen Tab gehörenden Elemente im :new(as Parameter dpassende as ::TabPage als Parent referenziert werden.

In der Klassendefinition muss die Methode für die Verwaltung der Tabs gesetzt werden:

```
      METHOD   ActivateTab
```

In der Init-Methode des Mutterdialogs muss der erste Tab als einziger sichtbar bleiben (mit Verweis auf die definierte Methode)::

```
   ::TabPage2:Minimized   := .T.
   ::TabPage1:TabActivate := { || ::ActivateTab( ::TabPage1 ) }
   ::TabPage2:TabActivate := { || ::ActivateTab( ::TabPage2 ) }
```

Die Servicefunktion selbst ist einfach:

```
   METHOD NewForm:ActivateTab( oTab )
   IF oTab:Minimized
      IF oTab == ::TabPage1
         ::TabPage2:Minimize()
         ::TabPage1:Maximize()
      ELSEIF oTab == ::TabPage2
         ::TabPage1:Minimize()
         ::TabPage2:Maximize()
```

```
      ENDIF
   ENDIF
   RETURN self
```

Bei mehreren Tabs wird entsprechend erweitert.

Achtung: Bei der Init() der einzelnen Tabpages müssen die Elemente, die in das entsprechende Tab gehören, ausschliesslich nach diesem Create() erstellt werden, sonst gibt's unliebsame Effekte oder gar den 4208-Fehler.

Im Formdesigner ist das Ganze unbrauchbar und unlogisch. Einzig für die Darstellung der Offsets der Reiter ist das Visuelle hilfreich.

Aktion aus XpbParts

```
oElement:activate := {|| funktion1, funktion2,...}.
```

Funktion1, 2 und... sind beliebige Funktionen eigene oder eingebaute.
Einfache Aufrufe können direkt in das activate geschrieben werden:

```
  : oButton:activate := {|| DBSkip()}
```

Lesen und Schreiben aus XpbParts

Dafür gibt es die internen Funktionen gather() und scatter(), denen ein Array mit Deinen Xbase++-Parts übergeben werden. Erstere schreibt in die Datenbank und Letztere liest aus der Datenbank.
Siehe dazu auch das Beispiel zu gather() in der Doku:

```
oXbp1 := XbpStatic():new( drawingArea, , {12,48}, {300,96} )
oXbp1:caption := "Customer"
oXbp1:type := XBPSTATIC_TYPE_GROUPBOX
oXbp1:create()

oXbp := XbpStatic():new( oXbp1, , {12,48}, {72,24} )
oXbp:caption := "Lastname:"
oXbp:options := XBPSTATIC_TEXT_VCENTER+XBPSTATIC_TEXT_RIGHT
oXbp:create()

oXbp := XbpSLE():new( oXbp1, , {96,48}, {192,24} )
oXbp:bufferLength := 20
oXbp:tabStop := .T.
oXbp:dataLink := {|x| IIf( PCount()==0, ;
                           Trim( CUSTOMER->LASTNAME ), ;
                           CUSTOMER->LASTNAME := x ) }

oXbp:create():setData()

AAdd( ::aEditControls, oXbp )

oXbp := XbpPushButton():new( drawingArea, , {12,12}, {72,24} )
oXbp:caption := "Vor"
oXbp:create()
oXbp:activate := {|| Gather( ::aEditControls ), ;
```

```
                    CUSTOMER->( DbSkip( 1) ), ;
                    Scatter( ::aEditControls ) }
oXbp:killInputFocus := { |x,y,oSLE| oSLE:getData() ,;   // Daten lesen
                                    do_other()          } // was machen
```

Gather() schreibt in die Datenbank.
Scatter() liest aus der Datenbank.
Bei Arbeiten mit Datenbankfeldern ist das TRIM(… sehr wichtig, weil sonst Leerzeichen in der SLE mitkommen (gemäss Bufferlength) und nur im Ueberschreibmodus Eingaben möglich wären.
Der Killinputfocus ist sehr gut, um etwelche Datenvalidierungen machen zu können. Dort wird auch der :datalink ausgeführt.
Was ist mit Spezialzeichen oder Verändern bestimmter Tasten (z.B. ESC)?
Jedem SLE einen keyboard-Slot zuweisen:

```
 bKeyHandler := {|nKey,uNIL,oObj| MainKeys(nKey,oObj) }
   AEval( aEditControls, {|o| o:keyBoard := bKeyHandler } )
```

Achtung: bei Readonly-Feldern reicht der Datalink in dieser Form aus:

```
 datalink := {|| xVar }
```

Der grosse Nachteil von gather ist, dass alle beteiligten Felder geblockt werden. Dies kann nicht immer gewünscht sein, vor allem, wenn der Benutzer in diesem Abfragezustand den Computer verlässt. Umgehung ist die Verwendung von Memvars (als Array) die mittels einer Funktion beim Schreibbefehl aktiviert wird.

```
oSLE:dataLink     := VAR1BLOCK(@cBegriff)
...
FUNCTION VAR1BLOCK(uMem)                                    // for SLEPIC
LOCAL bblock
RETURN {| x | IIF(x == NIL,uMem,uMem := x)}

FUNCTION VAR2BLOCK(uMem)                                    // for DBF Fields
LOCAL bblock
   DO CASE
      CASE VALTYPE(uMem) = "D"
         bblock := {| x | IIF(x == NIL,DTOC(uMem),uMem := CTOD(x))}
      CASE VALTYPE(uMem) = "C"
         bblock := {| x | IIF(x == NIL,uMem,uMem := x)}
      CASE VALTYPE(uMem) = "N"
         bblock := {| x | IIF(x == NIL,TRANSFORM(uMem,'@N'),uMem := VAL(x))}
      OTHERWISE
         bblock := {| x | IIF(x == NIL,uMem,uMem := x)}
   ENDCASE
RETURN bblock
```

Es geht auch ohne Gather und Datalink, indem man selber die Setdata() zum hineinschreiben der Werte in die SLE verwendet und zum Abfragen das GetData() beim Zeitpunkt des Speicherns abfragt.

Shortcut with Alt-

Die Alt-Taste ist in Windows seit jeher sehr wichtig. Sie dient in Menüs oder auch Dialogen zum direkten Springen auf denjenigen Dialogteil, welcher den unterstrichenen Buchstaben

(eben der mit der Alt-Taste) beinhaltet. Deshalb ist es wichtig, dass die Taborder stimmt, der Focus geht auf das nächstfolgende Inputfeld.

Bei den Menüs muss einzig der später mit ALT gewählte Buchstabe mit einem ~ vorgestellt werden.

Bei Dialogen ist keine Shortcutschlaufe implementiert. Glücklicherweise findet man aber unter den Samples das shortcut welches da Abhilfe schafft (source\solution\shortcut).

- Das scmanage.prg muss in das Projekt eingebunden werden
- Die Klasse muss erweitert werden um den Eintrag SCManager

```
CLASS NewForm FROM _NewForm, SCManager
```

- In derselben Klasse muss die Methode Keyboard definiert werden

```
METHOD keyboard
```

- **Die Methode selber natürlich auch**

```
METHOD NewForm:Keyboard( nKey)
   LOCAL lHandled := FALSE
   lHandled := ::HandleShortCut( nKey )
   IF ! lHandled
      RETURN ::xbpDialog:keyBoard( nKey )
   ENDIF
RETURN self
```

- Die Dialogtexte (Static müssen mit dem ~versehen werden

Bitmaps

Es gibt verschiedene Wege, Bitmaps zu laden

- aus einer Datei
- aus deiner Resourcen-Datei
-

Um Bitmaps in eine RES-Datei zu kriegen, muss eine Bilddatei in en ARC-File eingebunden werden.

Im .ARC-File so:

```
BITMAP  ID_ABOUT_BITMAP  =  "Xbpp.bmp"
```

In der Source oder im eigenen .CH so:

```
#INCLUDE ID_ABOUT_BITMAP   144
```

Der Resource Compiler erstellt aus dem .ARC und dem Bitmap eine .RES-Datei. Diese wird durch den Projektbuilder mit eingebunden.

Natürlich muss die ID_ABOUT_BITMAP bekannt sein - idealerweise als Nummer in einer Myapp.ch, sonst fehlt der logische Zusammenhang.

Jedem Dialogelement kann das Bitmap per Referenz zugeteilt werden.

```
   oDlg:drawingArea:bitmap := ID_ABOUT_BITMAP
```

Man kann auch das Bitmap als Objekt kreieren und dieses anschliessend so laden:

```
oBmp:= XbpBitmap():new()
oBmp:load(nil,2001)
oBmp:create(oDlg)

oXbp := XbpPushButton():new( drawingArea, , {500,200}, {230,190}, { {
XBP_PP_BGCLR, XBPSYSCLR_BUTTONMIDDLE }, { XBP_PP_FGCLR, -58 } } )
oXbp:caption := (oBmp)
oXbp:tabStop := .T.
oXbp:create()
```

Die Optionen können auch so geladen werden:

```
oXbp:drawingarea:options := …
```

Das Laden der Datei geht auch so:

```
oXbp:LoadFile(cBmp)
```

Achtung: Das Objekt der Bitmap bitte destroyen, sonst bleibt dies als Schrott im Speicher hängen.

```
oBmp:destroy()
```

Gruppen bilden

Es gibt verschiedene Wege, Gruppen zu bilden

Sollen Radiobuttons in einer Gruppe sein (stets einer darf nur aktiv sein), müssen diese in einer Gruppe lieben. Innerhalb der Gruppe gilt die Taborder.

Einzelne Elemente ansprechen

Einem Objekt kann eine eindeutige Zuweisung mittels SetName erfolgen.

```
oXbp:setName(ID_BTN_UNDO)
```

Später kann dieses Objekt direkt angesprochen werden:

```
oXbp := oDlg:ChildFromName(ID_BTN_UNDO)
oXbp:disable()
```

Ohne setname sieht Dasselbe so aus:

```
oDlg:XbpCheckBox:disable()
```

Cursor

Die Darstellung des Cursors kann erwünscht sein (Wait-Status usw.)

```
::DrawingArea:SetPointer(, XBPSTATIC_SYSICON_WAIT )
…
…
::DrawingArea:SetPointer(, XBPSTATIC_SYSICON_DEFAULT )
```

XbpParts

Wer kenn Sie nicht - die unverkennbaren Elemente auf den Dialogen: Pushbuttons (OK, Cancel), Radiobuttons, Listboxen, Eingabefenster usw.
Alle Elemente sind in Klassen definiert. Das heisst mittels Methoden und gegebenen Variablen werden diese Elemente erstellt und bewirtschaftet. Diese Instanzvariablen können auch Codeblöcke enthalten. Die Ereignisse, die die Ausführung abarbeiten, werden durch AppEvent() aus der Ereigniswarteschlange gelesen. Dies erfolgt im sog. CallBack-Verfahren.
Folgende Konstanten sind bei allen XbpParts vorhanden und erlauben eine Interaktion mit dem Benutzer (Inhalt ungleich NIL):

Ereignis-Code	Callback-Slot	Beschreibung
Mausereignisse		
xbeM_LbClick	:LbClick	Klick mit linker Taste
xbeM_LbDblClick	:LbDblClick	Doppelklick mit linker Taste
xbeM_LbDown	:LbDown	Linke Taste wurde gedrückt
xbeM_LbUp	:LbUp	Linke Taste wurde losgelassen
xbeM_MbClick	:MbClick	Klick mit mittlerer Taste
xbeM_MbDblClick	:MbDblClick	Doppelklick mit mittlerer Taste
xbeM_MbDown	:MbDown	Mittlere Taste wurde gedrückt
xbeM_MbUp	:MbUp	Mittlere Taste wurde losgelassen
xbeM_Motion	:Motion	Maus wurde bewegt
xbeM_RbClick	:RbClick	Klick mit rechter Taste
xbeM_RbDblClick	:RbDblClick	Doppelklick mit rechter Taste
xbeM_RbDown	:RbDown	Rechte Taste wurde gedrückt
xbeM_RbUp	:RbUp	Rechte Taste wurde losgelassen
Sonstige Ereignisse		
xbeP_Keyboard	:Keyboard	Tastatureingabe ist erfolgt
xbeP_HelpRequest	:HelpRequest	Hilfe anfordern
xbeP_SetInputFocus	:SetInputFocus	Input Fokus wurde erteilt
xbeP_KillInputFocus	:KillInputFocus	Input Fokus wurde entfernt
xbeP_Move	:Move	XBP wurde verschoben
xbeP_Paint	:Paint	XBP wurde neu gezeichnet
xbeP_Quit	:Quit	Anwendung wird abgebrochen
xbeP_Resize	:Resize	Größe eines XBPs wurde geändert

Die Elemente weisen auch eigene der Funktion des Parts passende CallBack-Slots auf.

Fenster erstellen

Im GUI lässt die Funktion XbpDialog() Fenster erstellen. Rahmen, Titel und Anzeigebereich gehören dazu. Alle Parts, die zum Fenster gehören, haben den Anzeigebereich als Parent (Instanzvariable :drawingarea).

Daten setzen

Element benennen / Label

Der Wert wird so gesetzt:

```
:MyObject:Caption := "Name"
```

Checkbox / Radiobutton

Der Wert einer Checkbox wird so gesetzt:

```
::MyCheckBox:State := 1
// Checked

::MyCheckBox:State := 0
// Unchecked
```

Achtung: Unter Vista muss bei ActiveX-Elementen IM FD unter Optionen/Einstellungen die Checkbox "Lizenzschlüssel ermitteln" aktiviert sein. Nur so wird die CLSID mit hineingeschrieben.

Wichtige CLSDI's:
Acrobat Reader: CA8A9780-280D-11CF-A24D-444553540000
Media Player: 6BF52A52-394A-11D3-B153-00C04F79FAA6

Lisbox / Combobox

Mit addItem werden die Elemente in die Liste aufgefüllt

```
  oComboBox:addItem("blabla")
usw…
```

Ist noch ein editierbares Eingabefeld bei der Box dabei, muss dessen Inhalt separat behandelt werden:

```
   cBuffer := oComboBox:XbpSLE:getData()
```

Es gibt ein Datenbankfeld FARBE vom Typ Zeichen
Der Datencodeblock verarbeitet eine Zeichenkette

```
  bData := {|c| IF( c != NIL, FIELD->FARBE := c, FIELD->FARBE ) }
```

Zuweisung des Datencodeblocks an den XbpSle-Teil der Combobox

```
  oCombobox:XbpSLE:dataLink := bData
```

Abfrage:

```
 ? ValType( oCombobox:GetData() ) // -> "C"
```

Ist kein editierbares Eingabefeld bei der Box dabei, reicht folgendes Vorgehen aus:
Es gibt ein Datenbankfeld PORT vom Typ Numerisch

Der Datencodebbilock verarbeitet ein Array

```
  bData := {|a| IF( Empty(a), FIELD->PORT := a[1], FIELD->PORT ) }
```

Abfrage:

```
 ? ValType( oCombobox:GetData() ) // -> "A"
```

Daten abfragen

Edit Element / Label

Der Titel eines Elements wird abgefragt mit:

```
cVar := ::Mylabel:Caption
```

Checkbox / Radiobutton

Abfragen einer Checkbox:

```
IF ::MyCheckBox:Checked()
// Checked
END
```

Abfragen mittels mp1

```
oXbp:selected := ;
       {| mp1, mp2, oChk| QOut( "Checkbox A", ;
                          IIf( mp1, "selected" , "not selected" ) ) }
```

Lisbox / Combobox

Abgefragt wird der Inhalt so, dass ein Array mit allen Optionen gefüllt wird.

```
  aItems := oComboBox:XbpListbox:getData()
```

Ist noch ein editierbares Eingabefeld bei der Box dabei, muss dessen Inhalt separat abgefragt werden, da dieser ja manipuliert werden kann:

```
   cBuffer := oComboBox:XbpSLE:getData()
```

Bei Multiselect-Listen können mehrere Elemente gewählt sein. Also müssen zuerst die dazugehörenden Indexe, anschliessend deren Inhalte in ein Array erfüllt werden:

```
  aItems := AEval( oLB:getData(), {|n| n:=oLB:getItem(n)}, , ,.T.)
```

Generell gibt das GetData den Index, nicht den Inhalt der gewählten Linien in einem Array an. Der Inhalt muss mittels GetItem abgefragt werden.

RTF-Daten

xbpRTF()

Prinzipiell dient dieses Part der Erstellung und Behabdkung von RTF-Dateien. Diese Dateien können in einem externen Programm (Wordpad, Office usw.) weiterverwendet werden.

```
oXbp := Xbprtf():new( drArea, , {345*nX, 56*nY}, {350*nX,380*nY}, { {
XBP_PP_BGCLR, XBPSYSCLR_ENTRYFIELD } } )
oXbp:create()
oXbp:textRTF:=hilfe->hilfe    // 'hilfe' ist ein Memo-Feld.
                              //  Hier könnte auch stehen
oXbp:LoadFile("Hilfe.RTF")
```

Wohin kommt mein eigener Code?

Schön, jetzt haben wir Fenster und Dialoge. Wo aber kommt nun unser eigener Code hin?

Bei den Klassen sind Funktionen in Methoden verfasst. So können wir unsere eigenen Funktionen natürlich auch als Methode definieren. Wichtig ist, wo diese dann ausgeführt werden.
Genau dazu verwenden wir die CallBack-Slots der XbpParts. Die Elemente dazu sind weiter oben beschrieben.

Active-X

Erscheint eher komplex - ermöglicht aber transparentes Arbeiten innerhalb verschiedner Applikationstypen.
Es braucht aber noch einige Dateien - gemäss Readme von xBase+:
:
Die Klassen XbpToolbar, XbpStatusbar, XbpRTF und XbpHTMLViewer
erfordern, daß verschiedene Systemkomponenten installiert sind.
Wenn Sie Fehler beim Erzeugen eines dieser Xbase Parts erhalten,
dann prüfen Sie bitte, ob Ihr System den dokumentierten System-
anforderungen des entsprechenden Xbase Parts entspricht. Ein
Installationsarchiv zum Installieren der benötigten System-
komponenten befindet sich im Verzeichnis \SETUP\RUNTIME\ENG
(Internationale Version) bzw. \SETUP\RUNTIME\GER (Deutsche
Version) auf Ihrer Produkt-CD.

Dazu muss die ocx-Datei zum registiert werden:

```
regsvr32 LW:\KOMPLETTER\PFAD\RichTx32.ocx
…
// copy X:\comctl32.ocx C:\WINDOWS\system32\
C:\WINDOWS\system32\regsvr32.exe comctl32.ocx
```

Aufrufen eines externen Programms

Aufruf externer Programme

Hier als Beispiel das Drucken eines Word-Dokuments:
Achtung: Kreierte Objekte müssen wieder zerstört werden, sonst bleibt die Speicherverwaltung nicht mehr geordnet.

```
    oWord := CreateObject("Word.Application")
```

```
   IF Empty( oWord )
     MsgBox( "Microsoft Word ist nicht installiert" )
     RETURN Nil
   ENDIF

   oWord:visible   := .f.                              // Word unsichtbar!
   cActivePrinter  := oWord:Activeprinter     // aktuellen Drucker feststellen
   oWord:Activeprinter:= cDrucker               // Neuen Drucker setzen
   oWord:documents:Open(cDoc)
   oDoc       := oWord:ActiveDocument
   oDoc:PrintOut(.f.)                                  // .T.= Druck im Hinter-
grund
   oWord:Activeprinter:= cActivePrinter       // alten Drucker setzen
   oDoc:close()
   oWord:Quit()
   oWord:destroy()
   oWord := NIL
```

Oder der Aufruf von Excel:

```
  // Create the "Excel.Application" object
  oExcel := CreateObject("Excel.Application")
  IF Empty( oExcel )
    tdMsg("Excel ist nicht installiert!" )
    RETURN NIL
  ENDIF
```

Der Befehl RunShell() ermöglicht das Aufrufen externer Programme.

```
RUNSHELL( "/C START MAILTO: /w" ,,.f.,.t.)
```

Runshell läuft über die Instanz des Command Interpreters ab. Deshalb wird kurz ein CmD-Fenster sichtbar sein, wenn eine Windows-Applikation gestartet wird.

```
oXbp:= XbpPushButton():new()
oXbp:caption := "Berechnen"
oXbp:create( , , {10,20}, {40,40} )
oXbp:activate := {|| //Programmstartanweisung }
```

Internet Explorer:

```
ShellExecuteA( AppDesktop():getHWnd(), SHELL_OPEN, "www.meineWebseite.de" , , ,
SW_NORMAL)
```

Allerdings muss die Funktion bekannt gegeben werden:

```
DLLFUNCTION ShellExecuteA( nWinHandle, cMode, cExe, cFile, cDir, nShow) USING
STDCALL FROM SHELL32.DLL
```

DLLFUNCTIONS:

```
DLLFUNCTION SetPixelV( hdc, x, y, crColor ) USING STDCALL FROM GDI32.DLL
DLLFUNCTION GetPixel( hdc, x, y ) USING STDCALL FROM GDI32.DLL
DLLFUNCTION GetWindowDC( hwnd ) USING STDCALL FROM USER32.DLL
DLLFUNCTION SendMessageA( nHwnd, nCmd, wParam, lParam ) USING STDCALL FROM US-
ER32.DLL
DLLFUNCTION ShellExecuteA( nHWND, cOperation, cFile, cParms, cDirectory,
nOpenMode  ) USING OSAPI FROM SHELL32.DLL
ort := textpfad + "\" +katalog + "\" + katalog + ".rtf"
ShellExecuteA( AppDesktop():GetHWND(), "open", ort, "", CurDir(), SW_NORMAL )
hWindow := DLLCALL( USER32DLL, DLL_STDCALL, "FindWindowA", 0, cName )
DLLCALL( "user32", 32,"MoveWindow", hWindow,100,100,1000,1000, .T.)
 LOCAL cBuffer     := Space(16)

   DllCall("User32.DLL", DLL_STDCALL,"GetWindowRect", hWindow, @cBuffer)

   nLeft    := Bin2U(substr(cBuffer,  1, 4))
   nTop     := Bin2U(substr(cBuffer,  5, 4))
   nRight   := Bin2U(substr(cBuffer,  9, 4))
   nBottom  := Bin2U(substr(cBuffer, 13, 4))
```

FUNCTION SUBDOCK(oParent)

```
...
   oDlg := XbpDialog():new( AppDesktop(),oParent, aPos, aSize, , .F.)
   oDlg:titlebar := .F.
   oDlg:border        := XBPDLG_SIZEBORDER      // will ReSize
   oDlg:moveWithOwner := .T.
   oDlg:drawingArea:bitmap := 1001
   oDlg:Resize := {|aOld,aNew,oSelf| myResize( aOld, aNew , oSelf)  }
   oDlg:create()
```

3rd Party Produkte

Verschiedene Fremdanbieter stellen Anwendungen oder Teile zur Verfügung, um das xBase+ attraktiver und praktischer zu gestalten.

Tx Text Control

Text Control bietet Word-Processing-Elemente an, die sich auch bei uns einbinden lassen.

```
oTxControl := XbpActiveXControl():new( oDlgArea )
oTxControl:CLSID := 'TIS.TX.TextControl.13'
oTxControl:license := 'haha,dieverrateichhiernatürlichnicht'
oTxControl:create(oDlgArea,,{0,hstatus},{w,htext} )
usw.
oTxControl:RbDown  := { |mp1, mp2, obj|oTxMenu:PopUp (oDlgArea, mp1 , 1 ) }
```

Das Menü dazu

```
oTxMenu := XbpMenu():new(oDlgArea):create()
oTxMenu:AddItem( {"Formatieren (Schrift)",{ || oTxCon-
trol:FontDialog(),SetAppFocus(oTxControl) } } )
oTxMenu:AddItem( {NIL, NIL, XBPMENUBAR_MIS_SEPARATOR, 0 } )
oTxMenu:AddItem( {"Löschen",{ || oTxControl:Clip(4) } } )
oTxMenu:AddItem( {"Ausschneiden",{ || oTxControl:Clip(1) } } )
oTxMenu:AddItem( {"Kopieren",{ || oTxControl:Clip(2) } } )
oTxMenu:AddItem( {"Einfü
```

Differenzen Xbase++ zu Clipper

Es gibt oft ärgerlich bei Konversionen diverse Unterschiede. Gleiche Funktionen aber anderes Verhalten.

APPEND FROM … SDF
Legt bei einem CRLF keinen neuen Datensatz mehr an.
Die Option NEXT liest ab Dateibeginn und nicht mehr ab aktuellem Datensatz ein. Dafür gibt es die Option RECORD.

CURDRIVE()
Xbase++ liefert nur den Buchstaben ohne Doppelpunkt!

FIELD(nPos)
Xbase++ kennt diese Anweisung per se nicht. **FIELDNAME(nPos)** verwenden!

Variablenname TO
Wird als reserved Keyword nicht mehr akzeptiert.

Dont's

Es gibt oft ärgerlich, dass der Compiler Fehler abgibt, die nicht sehr weiterhelfen.

XBase hat grösste Probleme, wenn in einer Methode oder Funktion ein ENDIF fehlt.

In Methoden nie die Variable END benutzen

Wunschliste

Allgemein

Form Designer

Absolute Scghwachstelle in XBASe++ - muss komplett ersetzt werden.
- Nachträgliches Edit wie bei den bekannten Resource Editoren
- Klasssen-Generierung ohne überschreiben
- Eigenschaften des Mutterfensters

Hierarchie der xBase-Parts

Class hierarchy of Xbase Parts

CLASS	DESCRIPTION
XbpPartHandler	Manages relationships and events
├─ XbpCrt	Hybrid window
├─ XbpHelp	Window for online help
├─ XbpSysWindow [A]	Abstract class for system dialogs
│ ├─ XbpFileDialog	File dialog
│ └─ XbpFontDialog	Font dialog
└─ XbpWindow [A]	Abstract class for windows
├─ XbpCellGroup	Cells in one column
│ └─ XbpMultiCellgroup	Cells in multiple columns
├─ XbpDialog	Dialog window
├─ XbpIWindow [A]	Abstract class for implicit windows
├─ XbpListBox (DataRef)	List box
├─ XbpMenuBar	Menu bar
│ └─ XbpMenu	Popup menu
├─ XbpMLE (DataRef)	Multiple line entry field
├─ XbpPushButton	Pushbutton
├─ XbpScrollBar (DataRef)	Scroll bar
├─ XbpSetting (DataRef) [A]	Abstract class for switches
│ ├─ Xbp3State	Three State Button
│ ├─ XbpCheckbox	Checkbox
│ └─ XbpRadioButton	Radiobutton
├─ XbpSLE (DataRef)	Single line entry field
│ ├─ XbpCombobox (XbpListBox)	Combo box
│ └─ XbpSpinButton	Spin button
├─ XbpStatic	Static dialog element
│ ├─ XbpBrowse (DataRef)	Browser
│ ├─ XbpQuickBrowse (DataRef)	Browser
│ └─ XbpColumn (DataRef)	Column for browser
├─ XbpTabPage	Tab page
└─ XbpTreeView (DataRef)	Tree view
XbpDevice [A]	Abstract class for device context
├─ XbpPrinter	Printer device context
└─ XbpFileDevice	Metafile device context
XbpFont	Fonts
XbpPresSpace	Presentation space
XbpBitmap	Bitmap
XbpMetaFile	Metafile
XbpHelpLabel	Help label

(...) = additional superclass, [A] = abstract class

Neue Funktionen in Xbase+

```
AppDesktop()        Devuelve la ventana del escritorio de Windows
AppEvent()          Lee eventos de la cola de eventos
AppKeyState()       Determina  el estado de teclas especiales
AppName()           Devuelve el nombre del EXE
AppType()           Determina el tipo de aplicación
Bin2f()             Convierte una cadena binaria a un numero FLOAT
Bin2u()             Convierte una cadena binaria a un entero LONG sin signo
Bin2Var()           Convierte una cadena binaria a cualquier tipo de dato
Char2Var()          Convierte una cadena binaria a otro tipo de dato
ClassCreate()       Crea una clase dinámicamente durante el tiempo de ejecución
ClassDestroy()      Libera una clase de objeto dinámica
ClassObject()       Devuelve la clase de objeto de cualquier clase
ConfirmBox()        Despliega la caja de diálogo GUI para confirmación del usuario
ConvToAnsiCP()      Convierte una cadena a ANSI
ConvToOemCP()       Convierte una cadena a OEM
CurDrive()          Devuelve la letra actual de la unidad
DispOutAt()         Salida de la pantalla sin cambio de la posición del cursor
DllCall()           Llama a una función contenida en un DLL
DllExecuteCall()  Llama una función DLL usando una plantilla de llamadas
DllInfo()           Devuelve información sobre uno o mas DLLs
DllLoad()           Carga un DLL dinámicamente durante el tiempo de ejecución
DllPrepareCall()  Prepara una plantilla de llamadas para una función DLL
DllUnLoad()         Descarga un DLL
EnableClipRect()  Permite el manejo de modo de pantalla de texto virtual
F2bin()             Convierte un numero FLOAT a binario
GetEnableEvents() Habilita al mouse para GETs
GetEventReader()  GETs usa AppEvent() en vez de Inkey()
GetHandleEvent()  Manejo de evento predeterminado para GETs
GetKillActive()  Toma foco de GET activo
GetList()           Devuelve el arreglo GetList actual
GetListPos()        Devuelve la posición del GET actual en GetList
GetToMousePos()  Mueve el cursor para el mouse dentro de GETs
IsFieldVar()        Chequea si existe el campo
IsFunction()        Chequea si existe función
IsMemberVar()       Chequea si el objeto tiene una variable de miembro específico
IsMemvar()          Chequea si existe la variable de memoria
IsMethod()          Chequea si el objeto tiene un método específico
LastAppEvent()      Devuelve el último evento
LChar2Var()         Convierte una cadena local a otro tipo de dato
MsgBox()            Despliega una caja de mensajes GUI
NextAppEvent()      Devuelve el próximo evento
NumButtons()        Numero de botones del mouse
PostAppEvent()      Envía un evento a la cola de eventos
PValue()            Recupera el valor del n-simo parámetro
RunRexx()           Ejecuta un archivo de comando REXX
RunShell()          Abre un nuevo interprete de comando
SetAppEvent()       Asocia un evento con un bloque de códigos
SetAppFocus()       Establece foco a una ventana o control GUI
SetAppWindow()      Devuelve la ventana de aplicación
SetClipRect()       Define el área de manejo para modo de texto
SetEnv()            Define una variable de entorno
SetLexRule()        Define reglas de comparación léxica para los caracteres
SetLocale()         Función para localización
```

```
SetMouse()          Permite los eventos del mouse en modo de texto
SetTimerEvent()  Inicia un temporizador de thread
Sleep()             Detiene el thread actual por un tiempo específico
SymbolInfo()        Devuelve información sobre símbolos de función
TBApplyKey()        Manipulador predeterminado  Inkey() para Tbrowse
TBHandleEvent()  Manipulador predeterminado AppEvent() para Tbrowse
TBtoMousePos()      Mueve el cursor de TBrowse al puntero del mouse
ThreadID()          Devuelve ID del thread actual
ThreadObject()      Devuelve el objeto Thread actual
ThreadWait()        Espera para que termine un thread
ThreadWaitAll()  Espera para que terminen múltiples threads
U2bin()             Convierte un entero LONG a binario
Var2Bin()           Convierte cualquier tipo de dato a binario
Var2Char()          Convierte cualquier tipo de dato a una cadena de caracteres
Var2LChar()         Convierte cualquier tipo de datos a una cadena local
```

Neue Funktionen im Bereich Datenbanken

```
DbCargo()               Añade un valor arbitrario a un área de trabajo usada
DbClientList()          Devuelve todos los clientes registrados de un área de trabajo
DbContinue()            Equivalente funcional de CONTINUE
DbCopyExtStruct()       Equivalente funcional de COPY STRUCTURE EXTENDED
DbCopyStruct()          Equivalente funcional de COPY STRUCTURE
DbCreateExtStruct()     Equivalente funcional de CREATE STRUCTURE EXTENDED
DbCreateFrom()          Equivalente funcional de CREATE FROM
DbDeregisterClient()    Remueve un cliente registrado de un área de trabajo
DbeBuild()              Construye un Motor de Base de Datos compuesto
DbeInfo()               Devuelve información sobre un Motor de Base de Datos
DbeList()               Devuelve los nombres de Motor de Base de Datos cargados
DbeLoad()               Carga un Motor de Base de Datos
DbeSetDefault()         Selecciona el Motor de Base de Datos predeterminado
DbeUnload()             Libera un Motor de Base de Datos de la memoria
DbExport()              Equivalente funcional de COPY TO
DbGoPosition()          Mueve el puntero de registros usando un valor porcentual
DbImport()              Equivalente funcional de APPEND FROM
DbInfo()                Devuelve información sobre un área de trabajo
DbJob()                 Asocia un área de trabajo con un bloque de códigos
DbJoin()                Equivalente funcional de JOIN
DbList()                Equivalente funcional de LIST
DbLocate()              Equivalente funcional de LOCATE
DbPack()                Equivalente funcional de PACK
DbPosition()        Devuelve la posición del puntero de registros como un valor porcentual
DbRegisterClient()      Registra un cliente en un área de trabajo
DbRelease()             Libera a un área de trabajo del espacio de trabajo actual
DbRequest()             Transfiere un área de trabajo a un espacio de trabajo actual
DbResetNotifications()  Permite notificaciones desde áreas de trabajo a los clientes
DbRList()               Devuelve una lista de relaciones
DbROrdName()            Devuelve el nombre de índice de una relación
DbSkipper()             Función skipper DBF predeterminada para TBrowse
DbSort()                Equivalente funcional de SORT
DbSuspendNotifications()Inhabilita notificaciones desde áreas de trabajo a clientes
DbTotal()               Equivalente funcional de TOTAL
DbUpdate()              Equivalente funcional de UPDATE
DbZap()                 Equivalente funcional de ZAP
FieldInfo()             Devuelve información del campo
OrdIsDescend()          Chequea si el índice esta descendiendo
OrdIsUnique()           Chequea si el índice es único
WorkSpaceEval()         Evalúa un bloque de códigos en todas las áreas de trabajo usadas
WorkSpaceList()         Devuelve el nombre de alias  de todas las áreas de trabajo usadas
```

Neue Direktiven, Deklarationen und Befehle

```
DacPagedDataStore() Provee cache de acceso a bases de datos
DacSession()         Establece conexiones con servidores de bases de datos
DataRef()            Clase para referenciar datos
Error()               Clase Error
Get()                 Clase Get
TBrowse()            Clase TBrowse
TbColumn()           Clase TBColumn
Thread()              Clase Thread
Signal()              Clase Signal
VCrt()                Clase Pantalla Virtual para modo de Texto
Xbp3State()          Clase Botón Tres Estados
XbpBitmap()          Clase Bitmap
XbpBrowse()          Clase GUI browser
XbpCheckBox()        Clase Casilla de Confirmación
XbpClipBoard()       Clase Papelero
XbpColumn()          Clase Columna para el Browser Gráfico
XbpComboBox()        Clase Combobox
XbpCrt()              Clase Ventana híbrida
XbpDialog()          Clase Ventana GUI
XbpFileDev()         Clase Dispositivo de archivo
XbpFileDialog()      Diálogo de selección de archivo
XbpFont()            Clase Fuente Tipográfico
XbpFontDialog()      Diálogo de selección de Fuente Tipográfico
XbpHelp()            Clase Ventana de Ayuda en Línea
XbpHelpLabel()       Clase Ayuda sensible de contexto
XbpIWindow()         Ventana implícita
XbpListBox()         Clase Lista Desplegable
XbpMenu()            Clase Menú
XbpMenuBar()         Clase Barra del Menú
XbpMetaFile()        Clase Metarchivo
XbpMLE()              Clase Edición Multi Línea
XbpPartHandler()     Clase Padre para Componentes Xbase
XbpPresSpace()       Espacio de Presentación
XbpPrinter()         Clase Impresora
XbpPrintDialog()     Clase Diálogo de Selección de la impresora
XbpPushButton()      Clase Botón
XbpQuickBrowse()     Clase GUI browser
XbpRadioButton()     Clase Botón
XbpScrollBar()       Clase Barra de desplazamiento
XbpSetting()         Clase Abstracta para parámetros
XbpSLE()              Clase Editor de una sola línea
XbpSpinButton()      Clase Botón Giratorio
XbpStatic()          Clase Control Estático GUI
XbpSysWindow()       Clase Abstracta para diálogos del sistema
XbpTabPage()         Clase Control de Solapas
XbpTreeView()        Clase Vista en Arbol
XbpTreeViewItem()  Elemento de una Vista en Arbol
XbpWindow()          Clase Abstracta de ventana
```

xBase+ Klassenfunktionen

```
#pragma          Permuta parámetros del compilador durante el tiempo de compila-
ción
ACCESS | ASSIGN Declara métodos de acceso/asignación
APPBROWSE        Parte de aplicación (GUI browser)
APPDISPLAY       Despliega partes de aplicación
APPEDIT          Parte de aplicación (ventana de Edición)
APPFIELD         Declara un campo para una parte de aplicación
CLASS            Declara una clase
CLASS METHOD     Declara un método de clase
CLASS VAR        Declara una variable de clase
DEFERRED         Declara un método como diferido
DLLFUNCTION      Crea una función que llama a una función DLL
FINAL            Declara un método como final
INLINE           Declara un método en línea
METHOD           Declara un método
SET CHARSET      Selecciona conjuntos de caracteres ANSI u OEM
SET COLLATION  Establece tabla de ordenamiento especifica de país
SET LEXICAL      Permite una comparación léxica de reglas para cadenas
SET TIME         Establece un formato de despliegue para el tiempo del sistema
SYNC              Declara un método como sincronizado
VAR               Declara una variable de instancia
```

Neue Funktionen der Grafikengine

```
GraArc()                 Dibuja un arco, circulo o elipses
GraBitBlt()              Copia un bitmap
GraBox()                 Dibuja un rectángulo
GraError()               Devuelve el último código de error
GraGetRGBIntensity()  Determina intensidades de color RGB
GraIsRGBColor()          Chequea si un valor de color es un color RGB
GraLine()                Dibuja una línea
GraMakeRGBColor()        Crea un valor de color RGB
GraMarker()              Dibuja un color de marcador
GraPathBegin()           Inicia un trazado gráfico
GraPathClip()            Recorta un trazo gráfico
GraPathEnd()             Cierra un trazado gráfico
GraPathFill()            Rellena un trazado gráfico
GraPathOutline()         Bordea un trazado gráfico
GraPos()                 Establece la posición del cursor gráfico
GraQueryTextBox()        Calcula las coordenadas para cadenas
GraRotate()              Rota la salida gráfica
GraScale()               Escala la salida gráfica
GraSegClose()            Cierra un segmento gráfico
GraSegDestroy()          Libera un segmento gráfico
GraSegDraw()             Dibuja un segmento gráfico
GraSegDrawMode()         Establece el modo dibujar para segmentos de gráfico
GraSegFind()             Encuentra segmentos gráficos
GraSegOpen()             Abre un segmento gráfico
GraSegPickResolution() Establece sensibilidad para hallar segmentos gráficos
GraSegPriority()         Establece prioridad para segmentos gráficos
GraSetAttrArea()         Establece atributos para áreas de dibujos
GraSetAttrLine()         Establece atributos para líneas de dibujo
GraSetAttrMarker()       Establece atributos para marcadores de dibujo
GraSetAttrString()       Establece atributos para cadenas gráficas
GraSetColor()            Establece el color para todas las funciones gráficas
GraSetFont()             Establece la fuente para cadenas gráficas
GraSpline()              Dibuja una curva
GraStringAt()            Dibuja una cadena
GraTranslate()           Desplaza la salida gráfica
```

DBU.EXE aus Clipper anpassen

```
*****************************************************************************
Changes for DBU.EXE 5.2e
*****************************************************************************

1. DBU.PRG

   o Rename PROCEDURE Dbu to PROCEDURE Main

2. DBUEDIT.PRG

   o Correct an implementation error which is tolerated by Clipper
     but not by Xbase++

     if ( "->" $ cEditField )
        cAlias := Substr(cEditField, 1, At("->", cEditField) + 1)
                This Plus must be a Minus ----------------^^^

   o Change two Inkey() calls to Inkey(0.1) for smooth operation

3. DBUVIEW.PRG

   o Move all "local saveColor" declarations in front of the
     PARAMETERS statement

   o Change one Inkey() to Inkey(0.1) for smooth operation

   o Change the code in FUNCTION bar_menu to reflect a slightly
     different behaviour of Xbase++'s Achoice() function.
     (Clipper's Achoice() does not display array elements which contain
      null strings, while Xbase++ does)

      FUNCTION bar_menu

      --------------------
      Old: local saveColor
      New: local saveColor, nElem

      --------------------------------
      Old: IF M->num_d < LEN(M->array)
              num_d = M->num_d + 1

              array[M->num_d] = " "
           ENDIF
```

```
New: nElem := LEN(M->array)
     IF M->num_d < nElem
        num_d = M->num_d + 1

        array[M->num_d] = " "
        ASize( m->array, M->num_d )
     ENDIF

-----------------------------
Old: IF array[M->num_d] == " "
        array[M->num_d] = ""
     ENDIF

New: IF array[M->num_d] == " "
        ASize( M->array, nElem )
        AFill( M->array, "", M->num_d )
     ENDIF
```

5. DBUUTIL.PRG

- Move all "local saveColor" declarations in front of the PARAMETERS statement

- Change this command with & to its functional equivalent (necessary because Xbase++ always uses late binding in code blocks)

```
Old:    SET RELATION ADDITIVE TO &k INTO &t

New:  DbSetRelation( t, &( "{||"+ k +"}" ), k )
```

- Change one Inkey() to Inkey(0.1) for smooth operation

6. DBUCOPY.PRG

- Move all "local saveColor" declarations in front of the PARAMETERS statement

****

Changes for RL.EXE 5.2e

****

1. RLFRONT.PRG

- Rename PROCEDURE SetUp() to Main()

- Add SET EXCLUSIVE ON in Main()

2. RLBACK.PRG

o Change FUNCTION LBL_LOAD to reflect additional operating system
error codes

Old: IF file_error = 2

New: IF file_error = 2 .OR. file_error = 110

3. RLDIALG.PRG

o Correct two wrong Achoice() calls in FUNCTION filelist

This line has an error:
achoice(wt+1,wl+32,wt+wh,wl+43,files,"ch_func",0,asel,arel)

This is correct:
achoice(wt+1,wl+32,wt+wh,wl+43,files,.T.,"ch_func",asel,arel)

(just search for "ch_func")

4. FRMDEF.CH

o Add one directive to the include file

#xtranslate IsAffirm(<c>) => (<c> $ "Yy")

5. LBLDEF.CH

o Add one directive to the include file

#xtranslate IsNegative(<c>) => (<c> $ "Nn")

6. STD.CH

o Replace Xbase++'s LABEL FORM and REPORT FORM #commands in STD.CH
with Clipper's #commands (Better: copy them into a separate
CH file and #include this file to PRGs which use the commands).

//EOF
/////Allgemein

Literaturverzeichnis

Alaska. (2006). *Alaska Xbase++*.
ALASKA Software. (kein Datum). Alaska Xbase++ Referenzdokumentation.
Inoffizielles deutsches Xbase-Forum. (kein Datum). Von www.xbaseforum.de abgerufen