KEY_TO_TRITON_dt

Philipp Lonke <phips@scout.franken.de>

COLLABORATORS

	TITLE : KEY_TO_TRITON_dt	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Philipp Lonke <phips@scout.franken.de>	February 11, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

KEY_TO_TRITON_dt

1.1 Programmierung eines TRITON-GUIs in Blitz2

```
                  TRITON - Schritt für Schritt
                  ----------------------------

Einführung              Worum es geht
Nützliches...           Grundwissen
Installation            Wie installiere ich das Packet?
Programmierbeispiel     "Hello world" in TRITON
Frage und Antwort       Problemlösungen
Zusätzliches            Sonstige Anmerkungen
Danksagungen            Zu guter Letzt.
Für Fortgeschrittene     Tips zur Optimierung

Neues                   Neues seit dem letzten Release

Über den Autor          Wie man an den tollen Typen
                         der das ganze verbrochen hat
                         rankommt :-)
Blitz-Kontakte          Wo gibt's was über Blitz2?
```

1.2 Einführung

```
Zuerst, vergeßt alles über die Blitz2-typische Art der
GUI-Programmierung. TRITON verfolgt ein ganz anderes Konzept, daß dem
von MUI ähnelt - aber auf keinem Fall der herkömmlichen Art eines
Gadtools-Guis.

Um die Unterschiede zu verstehen, sollten Sie diesen Guide sorgfältig
durchlesen. Bei Fragen reicht eine email an den Autor
Ebenso sollte die TRITON-Dokumentation sorgfältig durchgelesen werden.
```

1.3 Nützliche Definitionen

TRITON verwendet hauptsächlich zwei Ausdrücke:

application : Die Applikation ist das eigentliche Programm. Ihre Infos werden über Tags an den TRITON Preferences Editor (ShareWare) weitergegeben.

project : Das ist das eigentliche GUI. Jedes Fenster hat ein eigenes project mit eigener ID. Somit kann jede Applikation mehrere Projekte haben, aber nicht andersrum.

Desweiteren muß jeder Textstring, der an TRITON übergeben wird, mit CHR$(0) beendet sein. Seit Blitz1.9 sollte dies einfach durch den Befehl &string$ gehen. Ansonsten sollte man immer die Blitz2-Funktion Null("text") benutzen.

Eine ID von '0' (Null) darf *nicht* verwendet werden! Weder für die Applikation, noch für Projekte, Knöpfe etc.!

Wenn Sie das Programm mit dem Debugger unterbrechen, dann nicht das Programm über den Debugger beenden, sondern wieder auf "RUN" klicken und das Programm normal beenden. TRITON schließt sonst weder Fenster noch Applikation, was zu unangenehmen Abstürzen führen kann.

1.4 Installation der Konvertierung

Wichtig! Dieses Archiv ist wertlos ohne das TRITON Entwickler-Archiv. Dieses finden Sie im Aminet unter dev/gui/tritonXXdev.lha (XX ist die Versionsnummer und sollte mindestens 14 [also 1.4] sein!)
Den Preferences-Editor finden Sie im TRITON Benutzer-Archiv, ebenfalls im Aminet unter dev/gui/tritonXXusr.lha.
Der Prefs-Editor ist zwar Shareware, aber erlaubt dem Benutzer viele nützliche Einstellungen an den GUIs!

Da die triton.library ab diesem Release schon vorkompiliert ist, wird eigentlich nur noch das Benutzerarchiv benötigt - allerdings sind die Autodoc-Files nur im Developperarchiv enthalten!

Wenn Sie nicht wissen, welche Blitz-Library-IDs in ihrem System noch unbenutzt sind, sollten Sie entweder das Programm "Viewlibs" oder den neuen "LibMan" benutzen. Diese Programme sollten auf den Blitz2-FTP-sites zu finden sein.

Viewlibs: Einfach im CLI (oder Shell) "viewlibs" <return> eingeben, danach werden alle Libraries mit ihren IDs angezeigt. Nun müssen Sie nur noch zwei freie ID raussuchen - und merken.

Libman : LibMan von der Workbench starten, und die Ansicht zu "Sort by ID" schalten. Danach werden ebenfalls die Libraries mit ihren IDs angezeigt und Sie müssen nur noch zwei freie finden und merken.

Seit diesem Release haben die TRITONBLITZ-libraries eigene IDs und
sind schon vorkompiliert.

Achten Sie darauf, daß die IDs 219 (für TRITON), 26 (für TagListLib)
und 48 (für DBaseLib) frei sind! Diese IDs sind von RWE/Leading Edge
für diese drei Libraries festgelegt worden.

Kopieren sie tritonblitz/blitzlibs/amigalibs/triton.library1 in das
Verzeichnis Blitzlibs:amigalibs/

und .../userlibs/TagListLib.obj sowie DBaseLib.obj in das Verzeichnis
Blitzlibs:userlibs/

Nun müssen Sie ihr altes Deflibs-File löschen (oder besser ein
Backup aufbewahren!) und ein neues machen, entweder mit
"Makedeflibs" oder mit "LibMan".

Wählen Sie nun den Menupunkt "Reload all libs" (oder Beenden und
starten Sie Blitz2 erneut) und tippen

    TR_OpenProject_

ein. Dieser Befehl sollte nun die von ihnen eingestellte Tokenfarbe
annehmen. Bewegen Sie den Cursor auf den Befehl und drücken Sie
nun die HELP-Taste, der Hilfstext sollte in der Titelzeile
erscheinen.

Verfahren Sie ebenso mit dem Befehl

    InitTagList
und
    StrToFls

und drücken Sie wieder die HELP-Taste.

Sollten die Befehle nicht die Tokenfarbe annehmen oder kein
Hilfstext erscheinen, wiederholen Sie die Installation. Sollte
nach mehreren Versuchen und mehrmaligen Durchlesen des Guides
nichts funktionieren, kontaktieren Sie mich.
Eine Antwort kommt garantiert - und schnell.

Danach laden Sie das File "triton.bb2" in den Editor und speichern
es in ihrem Includes-Verzeichnis ab. Kopieren Sie es nicht direkt
dorthin, da es nicht tokenisiert ist!

1.5 Wie wird ein TRITON-GUI programmiert?

Die TagList-Library

TagListLib wurde ursprünglich von D. Pink geschrieben. Die Version 1.1
ist ein kleines Update, bei dem einige Fehler entfernt wurden und
andere Verbesserungen angebracht wurden. Der Autor der neuen Version
ist Patrik Rådman (pradman@mail.abo.fi 21-Jul-96)

Kurz einige Erläuterungen der Befehle:

o InitTagList TagList.w,NoTags.l
  * Besorgt den Speicher für eine Tagliste.

o AddTags [TagList.w] [[,Tag.l,Data.l]]
  * Die Angabe der Tagliste ist unnötig, wenn man diese vorher mit
    Use Taglist #nummer als aktuell deklariert hat.
    Es können beliebig viele Tag-Paare folgen. Eine Tagliste muß
    immer mit #TAG_END,0 enden (amigalibs.res dazuladen!)

o NoTagsLeft [TagList.w]
  * Gibt an, wieviele Tags noch frei sind in der Liste.

o TagList [TagList.w]
  * Gibt die Adresse der Tagliste im Speicher zurück.

(Ein kurzes Beispiel finden Sie entweder in den TRITONBlitz2-Sources
 oder in "Taglistlib_example.bb2")

Die DBaseLib

Diese Dokumentation ist etwas ausführlicher...

Die TRITON.library

Die Benutzung und Syntax der TRITON-Befehle lesen Sie bitte aufmerksam die Dokumentationen im Entwicklerarchiv von TRITON durch.

Sonstige Anmerkungen

Denken Sie immer daran, daß TRITON-Befehle System-Befehle sind.
Hängen Sie deshalb immer an den TRITON-Befehl einen Unterstrich "_" an.

Somit dürfen Sie nicht TR_OpenProject aber TR_OpenProject_ schreiben.

Wenn Sie in ihrem Programm mehr als 5 Taglisten verwenden wollen, erhöhen Sie das Maximum der TagListen in dem "Compiler-Options"-Fenster.

Endlich geht’s los

Zuerst müssen das Includefile geladen werden (amigalib.res in Compiler-Options bei "Residents" eintragen!!) und die Taglisten initialisiert werden.

```
    INCLUDE "blitz2:bbincludes/libraries/TRITON.bb2"    ; Oder wo es sich
                                                        ; sonst befindet
    InitTaglist 2,10                    ; Platz für die Applikation
    InitTagList 1,20                    ; Platz für 10 Tag-Paare
                                        ; in Liste 1

    Use TagList 2                       ; Und die Applikationsliste
                                        ; aktuell machen
```

Bevor wir TRITON mitteilen, daß wir eine neue Applikation starten, müssen wir noch ein paar Infos vorbereiten. Diese Infos sind nur für den Prefs-Editor von TRITON wichtig.

Die Tags dafür sind:

```
#TRCA_Name
#TRCA_LongName
#TRCA_Info
#TRCA_Version
#TRCA_Release
#TRCA_Date
```

Und im Source sieht das dann so aus:

```
AddTags #TRCA_Name,Null("TritonTemplate")
AddTags #TRCA_LongName,Null("TritonTemplate")
AddTags #TRCA_Info,Null("Looks like a template")
AddTags #TAG_END,0
```

Nunn können wir die Applikation öffnen. Immer daran denken, daß die Variable vom Typ Long sein muß!

```
application.l=TR_CreateApp_ (TagList)
```

Dann wird geprüft, daß alles geklappt hat:

```
IF (application)

    ... code ...

Else

    ... Fehler-Code ....

endif
```

Nun kommt endlich das GUI dran. Als Beispiel nehmen wir ein Fenster mit zwei Knöpfen. Wir wechseln nun zu TagList 1 für das Layout.

```
Use TagList 1
```

Das Layout läßt sich am einfachsten mit den Macros aus "triton.bb2" programmieren.

Um ein Fenster zu schaffen, sind die wichtigsten:

```
!WindowID{id}   - `id` darf niemals den Wert Null haben!

!WindowPosition{positionflag}

!WindowTitle{Null("Window title")}

!WindowFlags{flag1|flag2|flag3|...}
```

Nun kommen wir zu den Knöpfen. Knöpfe, wie alle anderen Elemente, müssen für TRITON gruppiert werden, entweder horizontal oder

vertikal. Das ist wichtig, damit TRITON das Layout richtig berechnen kann.

```
        Gruppen : !HorizGroup, !VertGroup
                    (Was die Gruppen bedeuten, steht in der
                     TRITON-Dokumentation.
                     Eine muss immer mit dem Macro !EndGroup
                     geschlossen werden. Sämtliche TRITON-GUI-Elemente
                     müssen gruppiert werden!!!)

        Knöpfe  : !Button{Null("T_ext"),id}
                    (Die sog. hot keys, mit denen der Knopf per Tastatur
                    ausgelöst werden kann, gibt man durch einen Unter-
                    strich vor dem Buchstaben an [in diesem Falle "e"])
```

Im Source sieht das Ganze dann so aus:

```
        AddTags !VertGroupA
        AddTags     !Space
        AddTags     !HorizGroupA
        AddTags         !Space
        AddTags         !Button{Null("_Save"),1}
        AddTags         !Button{Null("_Cancel"),2}
        AddTags         !Space
        AddTags     !EndGroup
        AddTags     !Space
        AddTags !EndGroup
        AddTags #TAG_END,0
```

Dran denken, eine Taglist muß mit #TAG_END,0 enden, sonst gibt’s Verwirrung!

Die obige strukturierte Programmierung lohnt durchaus - man behält leichter den Überblick...

Nun können wir endlich das Fenster bzw. Projekt öffnen. Dies geschieht wieder über eine Variable vom Typ Long.

```
        project.l=TR_OpenProject_(application,TagList)
```

Und nun das eigentliche Programm:

```
        if (project)   ; Wenn alles geklappt hat

            close_me.b=False      ; merken: Schließgadget nicht gedrückt

            while NOT close_me    ; und solange das so bleibt
                TR_Wait_ application,0     ; warten wir auf Nachricht von ↩
                   TRITON

                *trmsg.TR_Message=TR_GetMsg_(application)
                                  ; was macht der User?

                while (*trmsg)
```

```
                if (*trmsg\trm_Project=project)    ; Ist die Nachricht
                                                   ; für unser Projekt?

                  select *trmsg\trm_Class       ; Welche Klasse der  ↩
                     Nachricht?

                    case #TRMS_CLOSEWINDOW
                        close_me=True           ; Schließgadget gedrückt

                    case #TRMS_ACTION           ; einen Knopf gedrückt

                      select *trmsg\trm_ID      ; aber welchen?

                        case 1
                            ; button 1
                            ; code

                        case 2
                            ; button 2
                            ; code

                      end select

                    case #TRMS_NEWVALUE     ; Elemente senden entweder  ↩
                       eine
                                            ; #TRMS_ACTION oder ... ↩
                                               NEWVALUE
                                            ; Nachricht zurück.
                                            ; NEWVALUE kommt zB. von  ↩
                                               Checkboxen
                                            ; und ListViews
                  end select
                endif

                TR_ReplyMsg_ *trmsg     ; Eine Nachricht immer so schnell
                                        ; wie möglich beantworten, damit  ↩
                                           TRITON
                                        ; die nächste senden kann.

                *trmsg=TR_GetMsg_ (application) ; Dann holen wir die  ↩
                   nächste ab

            wend
        wend

      TR_CloseProject_ project         ; Fenster schließen
    else
        NPrint "Unable to create project"       ; Wenn's nicht geklappt  ↩
           hat
    endif

    TR_DeleteApp_ application          ; Und TRITON mitteilen, daß
                                       ; die Applikation auch zu ende ist
else

    nprint "unable to create application"   ; bei Problemen
```

```
     endif

     end
```

Ein Blick direkt in das Includefile lohnt sich auf jeden Fall, um alle Gadgets etc. kennenzulernen.

Wie man die QuickHelp-Funktion von TRITON verwendet, kann man sehr gut aus dem Listing von TOOLMANAGER1a.bb2 ersehen.

Anstelle des Umstaendlichen Weges, die Listview-Liste über ein Array zu füllen, kann man auch die normale Blitz2-Listen benutzen. Wie das geht, erfahren Sie hier.

Ebenso bietet TRITON einen Systemrequester.

1.6 TRITON ListViews

Hier ein Ausschnitt aus dem Listing von TOOLMANAGER1.bb2 zur Verwendung der Blitz2-Listen in TRITON-Listviews.

```
; ... start code snipped ....

NEWTYPE .LVItem             ; dieser Newtype muß sein!
                            ; (vgl. Gadtools)
  num.w
  text$

End NEWTYPE

Dim List LVNodes.LVItem(9)

InitTagList 1,200

If AddItem(LVNodes())
  LVNodes()\text="2024View"
  If AddItem(LVNodes())
    LVNodes()\text="Add to archive"
    If AddItem(LVNodes())
      LVNodes()\text="Deletetool"
      If AddItem(LVNodes())
        LVNodes()\text="Edit text"
        If AddItem(LVNodes())
          LVNodes()\text="Env"
          If AddItem(LVNodes())
            LVNodes()\text="Exchange"
            If AddItem(LVNodes())
              LVNodes()\text="Multiview"
            EndIf
          EndIf
        EndIf
```

```
      EndIf
    EndIf
  EndIf
EndIf

ResetList LVNodes()

; ... application tags snipped ...

Use TagList 1

; ... rest of gui snipped ....

AddTags   !HorizGroupAC
AddTags     !Space
AddTags     !VertGroupAC
AddTags       !CenteredTextID{Null("Object List"),2}
AddTags       !Space
AddTags       !ListSSCN{&LVNodes(0)-36,2,0,0}
AddTags     !EndGroup
```

Wobei es zwar etwas unorthodox ist, auf ein Listenarray direkt zuzugreifen (vgl. Blitz2-Handbuch), aber es funktioniert ohne Abstürze.

Eine sauberere Methode bietet die DBaseLib von Graham .A. Kennedy (gakennedy@cix.compulink.co.uk). Sie ermöglicht zum einen ein besseres Hinzufügen/Löschen von Listenelementen und die Liste hat keine Fixgröße, sondern kann beliebig erweitert werden.

Hier ein Beispiel, auch zu sehen in toolmanager1a.bb2

```
NEWTYPE .LVItem
  text.b[20]
End NEWTYPE

#text = 20    ; maxlen for the LV-Text-Lines.

DEFTYPE .LVItem LVNodes

ok.b=DBInit (1,1,1,LVNodes,20)      ; initialize Database to LV-Text

If ok<>1                            ; if it fails, stop the program
  r=Request("Error","Could not create database","End")
  End
EndIf

;   .....

StrToFls "2024View",LVNodes\text,#text      : DBadd 1,LVNodes
StrToFls "Add to archive",LVNodes\text,#text: DBadd 1,LVNodes
StrToFls "DeleteTool",LVNodes\text,#text    : DBadd 1,LVNodes
StrToFls "Edit text",LVNodes\text,#text     : DBadd 1,LVNodes
StrToFls "Env",LVNodes\text,#text           : DBadd 1,LVNodes
StrToFls "Exchange",LVNodes\text,#text      : DBadd 1,LVNodes
```

```
StrToFls "Multiview",LVNodes\text,#text     : DBadd 1,LVNodes

; .....

AddTags   !HorizGroupAC
AddTags     !Space
AddTags     !VertGroupAC
AddTags       !CenteredTextID{Null("Object List"),2}
AddTags       !Space
AddTags       !ListSSCN{DBlistaddr(1),2,0,0}
AddTags     !EndGroup
```

1.7 Positionflags

Mögliche Flags sind:

```
#TRWP_DEFAULT
#TRWP_BELOWTITLEBAR
#TRWP_CENTERTOP
#TRWP_TOPLEFTSCREEN
#TRWP_CENTERSCREEN
#TRWP_CENTERDISPLAY
#TRWP_MOUSEPOINTER
#TRWP_ABOVECOORDS
#TRWP_BELOWCOORDS
```

1.8 Windowflags

Diese Flags können mit "OR" bzw. "|" miteinander kombiniert werden.

```
#TRWF_BACKDROP
#TRWF_NODRAGBAR
#TRWF_NODEPTHGADGET
#TRWF_NOCLOSEGADGET
#TRWF_NOACTIVATE
#TRWF_NOESCCLOSE
#TRWF_NOPSCRFALLBACK
#TRWF_NOZIPGADGET
#TRWF_ZIPCENTERTOP
#TRWF_NOMINTEXTWIDTH
#TRWF_NOSIZEGADGET
#TRWF_NOFONTFALLBACK
#TRWF_NODELZIP
#TRWF_SIMPLEREFRESH
#TRWF_ZIPTOCURRENTPOS
#TRWF_APPWINDOW
#TRWF_ACTIVATESTRGAD
#TRWF_HELP
#TRWF_SYSTEMACTION
```

1.9 TRITON's easyrequester

```
Die Funktion TR_EasyRequest_ (app.l,Null(body$),Null(gad$),tags)
läßt einen Systemrequester erscheinen.

Es gibt zwei Möglichkeiten, ihn zu verwenden:

    Bei einem einzigen Gadget

    TR_EasyRequest_ application.l,Null("Fehler!"),Null("Ok"),tags

    Bei mehreren Gadgets

    result.b=TR_EasyRequest_ (app.l,Null("Ende?"),Null("Ja|Nein"),tags)

    Wobei das erste, linke Gadget den Wert Null hat. Ansonsten wird
    beim letzten, rechten Gadget mit "1" nach links gezählt.

Mögliche Tags sind:

       #TREZ_ReqPos,position   : use the #TRWP_ tags for
                                positioning the requester
       #TREZ_LockProject,bool  : should the requester
                                lock the project? True or
                                false, so you needn't use
                                TR_LockProject_ every time
       #TREZ_Return
       #TREZ_Title,Null("Title")
       #TREZ_Activate,bool
```

1.10 Frequently asked questions

```
Q: Wie gehe ich mit Listviews um?

A: Das steht hier

Q: Wie kann ich den Inhalt eines Listviews verändern?

A: Ganz einfach:

    TR_SetAttribute_ project,id,0,-1

    ; hier nun die Liste verändern

    TR_SetAttribute_ project,id,0,&List(0)-36

    ; oder, bei verwendung der DBaseLib

    TR_SetAttribute_ project,id,0,DBListAddr(#num_list)

 Q: In meinem GUI ist ein ReturnOK-Knopf und ein Stringgadget.
    Immer, wenn ich nun etwas im Stringgadget eingebe und mit Return
    beende, wird der Knopf gedrückt. Wie kann ich das vermeiden?
```

A: Einfach das Macro !StringGadgetNR anstelle von!_StringGadget
verwenden.

Q: Wie kann ich einen Zeichensatz mit unveränderbarer Breite
verwenden?

A: Ganz einfach. Bei den Windowtags im Layout folgendes hinzufügen:

```
AddTags #TRWI_FixedWidthFontAttr,font
```

Wobei font als font.TextAttr=Null("name.font"),groesse
initialisiert werden muß.

Gadgets, die diesen festen Zeichensatz verwenden sollen, beginnen
mit !FW. Am besten mal im Include-File nachschauen!

Q: Wie komme ich an den Inhalt eines Stringgadgets?

A: Sie müssen zu dem entspr. Zeiger ein Peek$ machen:

```
*text=TR_GetAttribute_(project,stringID,0)
text$=peek$(*text)
```

Q: Wenn ich das Macro !ListROC{...} in meiner Tagliste
verwende, bekomme ich beim Compilieren eine Fehlermeldung.

A: Das Macro !ListROC{} darf nicht hinter einem AddTags-Befehl
stehen, da es selbst diesen Befehl enthält.

```
; Beispiel zum verwenden des Macros !ListROC{}

AddTags !VertGroupAC
AddTags   !Button{Null("Text"),id1}
          !ListROC{DBListAddr(0),id2,0}
AddTags   !Button{Null("Text 2"),id3}
AddTags !EndGroup
```

1.11 Some final words

Ich hoffe, daß der Unterschied zwischen dem Erstellen und Behandeln
eines TRITON GUIs und eines GadTools (oder schlimmer, eines original
Blitz-)GUIs verstanden wurde.

Von jetzt an müssen Sie keinen Gedanken mehr an Fontsensitivity,
Layout, Fenstervergrößerungen etc. verschwenden - TRITON macht
das alles für Sie.

Benutzen Sie die Demo-Listings und diesen Guide als ein Beispiel,
wie man eigene GUIs programmiert. Beachten Sie aber immer diese
Regeln:

a) Verwenden Sie goto und/oder gosub nur selten in Ihrem
   Programm. Es ist ein schlechter Stil, vor allem, da
   Blitz2 die tolle Möglichkeit der Functions und Statements
   bietet!

```
    b) Jedes Macro, das den gleichen Namen wie ein Blitz2-
       Befehl hat, beginnt mit einem Unterstrich "_".
       Sollte also Ihr Macrobefehl die Tokenfarbe annehmen,
       einfach entspr. ändern.

    c) Bevor man die Nachricht von TRITON abholt, sollte man
       immer TR_Wait_ app.l,0 verwenden.

    d) Immer überprüfen, ob die Applikation/das Projekt
       auch geöffnet wurden!

    e) Jede Nachricht von TRITON so schnell wie möglich beantworten.

    f) Die Macros sind der einfachste Weg, das TRITON-GUI zu
       entwerfen. Es sind aber so viele, daß ich sie in diesem
       Guide nicht erklären kann.
       Die Namen sind aber schon so selbsterklärend, daß ein
       Blick in das Include-File auf jeden Fall ausreichen sollte.

Vor allem: Wegen des TRITON Preferences Editor kann der Benutzer
nicht nur das Aussehen der Gadgets, sondern auch die Größe des
Fensters, den Bildschirm und vieles mehr bestimmen.
Benutzen Sie deshalb niemals fixe Koordinaten, Bildschirmnamen
oder ähnliches, was Sie vielleicht vom vorherigen programmieren
in Blitz2 kennen.

Für Probleme:
         Authors adress
```

1.12 Kontakt mit dem Autor

```
Vorschläge, Verbesserungen, Anregungen, Sorgen, Nöte, Anträge,
Hilfesuche oder sonstiges gibt es bei:

    via eMail:  phips@scout.franken.de

                in der BlitzBasic-Mailingliste
```

1.13 Danksagungen und mehr...

```
Danke möchte ich all' diesen Leuten sagen (ohne Reihenfolge)

    · Rupert Henson für die Hilfe bei den Macros
    · D.C.J. Pink für seine TagListLib und
    · Patrik Rådman für die Verbesserungen
    · Stefan Zeiger für TRITON
    · Graham Kennedy für die DBaseLib
    · ACID Software für BlitzBasic2
    ·~Irena, meiner Freundin, für alles
    · Michael Bergmann für lustige Anrufe, wilde Parties,
      und Gesprächen über Computer in jeder (un-)möglichen
```

```
      Ecke der Welt.
     ·~James Savage seine Versuche, TRITON zum Funktionieren
      zu bringen
     · Stefan Haefner dafür, daß er mir die Wichtigkeit eines
      deutschen Guidefiles aufzeigte
     · Graham Kennedy für seine Tips
     ·~Falk Nieder für die Blitz2-Station in Deutschland
     · und allen anderen, die erwähnt werden sollten.

BlitzBasic2 ist (c) von ACID Software    <acid@iconz.co.nz>
TagListLib.bb2 ist (c) von D.C.J. Pink   <danpink@danpink.demon.co.uk>
Triton ist (c) von Stefan Zeiger         <s.zeiger@isobel.rhein-main.de>
DBaseLib ist (c) Graham Kennedy          <gakennedy@cix.compulink.co.uk>

TagListLib.bb2 ist Freeware. Triton ist Shareware. DBaseLib ist Freeware

Legales: DIESES ARCHIV IST FREELY DISTRIBUTABLE SOFTWARE UND
PUBLIC DOMAIN. ICH ÜBERNEHME KEINE GARANTIE FÜR IRGENDETWAS,
WAS IHNEN ODER IHREM COMPUTER BEI DER ANWENDUNG DER PROGRAMME
PASSIERT. DER SOURCECODE KANN VERÄNDERT WERDEN UND WEITERGEGEBEN,
SOLANGE ER IM ZUSAMMENHANG MIT DEM ARCHIV BLEIBT UND MEIN NAME
SOWIE DIE DER ANDEREN ERWÄHNTEN PD/FD-PROGRAMMIERER NICHT GELÖSCHT
ODER VERÄNDERT WERDEN.
ICH BITTE DARUM, JEDE VERÄNDERUNG MIR MITZUTEILEN, DAMIT ICH SIE
EVTL. IN ZUKÜNFTIGEN UPDATES EINSCHLIESSEN KANN.

keep on blizzing,

        phips@scout.franken.de
```

1.14 About TRITON

```
            ********************************************

                              Triton

                  An object oriented GUI creation system.

                        (c) 1993-1995 Stefan Zeiger

            ********************************************
```

Triton is an object oriented GUI creation system for AmigaOS. Triton makes it much easier to create good looking graphical user interfaces (GUIs) than GadTools, BOOPSI or other systems. Complicated things like resizability of windows or a fully font sensitive gadget layout are handled entirely by Triton.

Furthermore Triton GUIs can be configured by means of a Preferences editor, including e.g. a screen and a window manager for most comfortable GUI management.

There is a mailing list for discussions and questions about Triton.

```
If you have any problems with Triton or simply want to get in touch
with other developers who are using Triton, you can subscribe to the
list.

   In that case, send EMail to majordomo@mail.im.net with any subject
and the line 'subscribe triton' in the body of your message. If you
want the list mail to be sent to a different EMail address (and *only*
if you want this), please use 'subscribe triton
a.different@email.address' instead (after replacing
'a.different@email.address' with the address to send the mail to of
course).

   In order to unsubscribe from the list, simply follow the above
rules, replacing 'subscribe' by 'unsubscribe'.

   If you need more help, send mail to majordomo@mail.im.net with a line
'help' in the body.
```

1.15 Über BlitzBasic2

```
    BlitzBasic 2.1 ist (c) von Acid Software
    LibMan is (c) BlitzBasic Distribution Köln und
              geschrieben by Peter Eisenlohr

    Macht bei der BlitzBasic-Mailingliste mit:

         To: blitz-list-request@netsoc.ucd.ie
         Subject: help

    Ihr bekommt dann eine Mail, die weitere Auskünfte zur Anmeldung
    bei der Blitz2-Mailingliste gibt.

    Oder meldet euch in der deutschsprachigen Blitz2-Liste an:

        To: blitzlist@timewarp.insider.org
        Subject: subscribe

    BlitzBasic-FTP gibt es hier:

        x2ftp.oulu.fi/pub/amgiga/prog/blitz
        ftp.thenet.co.uk/users/hawkftp/developer/blitz

        oder im Aminet unter dev/basic

    Deutsche BlitzBasic-BBS

        0221-9320931 (ISDN-v34)
```

1.16 Für fortgeschrittene Programmierer

Es gibt eine Möglichkeit, die ausführbaren Programme ein ganzes
Stück kleiner zu machen.

Zuerst sollte der Gebrauch von Null(s$) eingeschränkt werden, oder
es gar nicht verwendet werden. Ab Blitz2 v1.9 funktioniert die
Übergabe der Strings mit dem Adress-Befehl genauso

```
    &string$ = Null(string$)
```

Dieser String darf aber nicht verändert werden, solange er noch
nicht an TRITON übergeben wurde!

Ebenso kann man die Strings als Labels am Ende des Sources
eingeben und sich im Befehl auf sie beziehen:

```
... ?string1

string1:
dc.b "Testtext",0
Even
```

Wie das genau funktioniert, kann man aus dem Listing von
"Memo2.bb2" von D. Pink ersehen.

1.17 Änderungen

1.0

Tippfehler beseitig.

Macro !ListROC funktioniert nicht

2.0

Durch die Taglist-Library konnte das Macro !ListROC so verändert
werden, dass es jetzt funktioniert. Anwendung siehe Index

2.1

Mit deutscher Uebersetzung (im BUM 9)

2.2

Die DBaseLib kommt zum Packet dazu.
Alle Libraries (triton,TagList,DBase) sind ab jetzt vorkompiliert
und haben offizielle IDs von RWE.

1.18 Anleitung zur DBaseLib

Diese Dokumentation liegt momentan nur in Englisch vor, und ich
werde sie in nächster Zeit auch nicht übersetzen können.

```
    Das File ist ebenso im Verzeichnis tritonblitz/blitzlibs/userlibs/
    vorhanden.

Database Function Library
Graham .A. Kennedy (gakennedy@cix.compulink.co.uk)
Version: 1.0  (20/02/95)
Library Number : 10 (needs real number defining)


------------------- Database Library Documentation ---------------------

Introduction:
  This library is provided to supply Blitz Basic with a number of simple
Database functions, which may be used either, obviously within a database
application (eg. the enclosed Address book program) or any program which
needs an array which can expand upto the size of the free memory available.
It also includes a number of functions which may be of use to anyone
wishing to use fixed length strings within a newtype.

Concepts:
  Some of the features of the database probably need some additional
description before we launch straight into the command syntax, so here
we go... hope it's not too boring...

Database structure -
  The database is controlled by a Blitz Object, which can be accessed
from the compiler options requester. Initially the number of databases
is set to 16, but this can be increased (or decreased) depending on your
requirements.

  The database itself is stored as a standard Amiga Exec name list, and
uses the internal internal functions to create, add and remove entries.

A database may be 'KEYED' ie. all or part of the record could be used as
a key. If you add data to a keyed file it will be inserted in key order.
Therefore, a keyed database is automatically in ascending order, and
requires no sorting.

Fixed length strings -
 These are a bit of a cludge to get around the fact that Blitz only stores
a pointer to a string inside a newtype variable. Storing a string within
the newtype itself, allows allsorts of interesting tricks, such as passing
a database to a GTListview gadget to display, or saving a whole newtype to
disk with one command. They are implemented by using a byte array within
the newtype. eg.

  NewType.mytype
    name.b[30]:;  Fixed length string 30 characters in length
    addrs.b[60]:;    "      "     "    60     "       "    "
    age.l
  end newtype

  deftype.mytype test:; make test a variable of mytype.

I will use this example when trying to describe the function of some
of the following commands.
```

```
***** N O T E *****
Please not you CANNOT use standard strings within a database Newtype.

Known Bugs:
  As far as I can tell there is only one known bug, which I hope will not
  be too much of a problem.
  If a database is filled so that it automatically expands, the total size
  of the new database is then used whenever the database is reloaded from
  disk, therefore taking up more memory than actually required.
  eg. a database is created with 500 records, and expands by 100 records
  each time it fills up. If 700 records are added then 600 deleted (leaving
  100 records in the database). Whenever this database is saved to disk and
  reloaded it will allocate space for 700 records when reloading, even though
  only 100 records are loaded).

  So far I havn't found too many problems with this situation, I do know how
  to fix this, and probably will if anyone thinks the library is worthwhile.

------------------------------ Command Documentation ---------------------


Database Commands:

Statement     : StrToFls

Syntax        : StrToFls string$,flspointer,length[,padchar]

Description   : This allows you to set a fixed length string to a value
                contained in a string. If the string is shorter than the
                length requested the fixed length string will be padded
                using the character defined by 'padchar'. If 'padchar' is
                ommitted '0' is used. If the string is longer than 'length',
                only 'length' bytes will be copied.

Example       : ; copies "Joe Bloggs" to field \name in test variable and
                ; pads field with spaces.
                a$="Joe Bloggs"
                StrToFls a$,test\name,30,32

                            -----------------

Function      : FlsToStr

Syntax        : ret$=FlsToStr$(flspointer,length)

Description   : This allows you to convert a Fixed length string to a
                standard Blitz string. The string created is returned in
                ret$. The string will be copied either until the first '0'
                byte is found or 'length' bytes have been copied.

Example       : ; Copies "Joe Bloggs" back to a$
                a$=FlsToStr$(test\name,30)

                            -----------------

Function      : DBinit
```

```
Syntax        : ret.b=DBinit(db#,primary,secondary,recvar[,keylen[,offset]])

Description   : This command initialises and builds a database, if the
                database is already in use it will be destroyed and a
                new one created. If the database is created the function
                will return 1, if it fails it will return 0.
                db#       = Database number
                primary   = Number of record initially allocated to database
                secondary = Number of record to add if database fills up
                recvar    = variable to use to define record structure
                keylen    = key database on this number of bytes
                offset    = Offset the key this number of bytes from the
                            start of the record.

Example       : ; define database number 1, give it space for 100 records
                ; initially, and expand the database by 10 records each time
                ; it fills up. Use our example newtype to define its structure
                ; and key it on the name field.
                  ret=DBinit{1,100,10,test,30)
                  if ret=1 then Nprint "Yippee, database defined"

                            -----------------

Function      : DBlistaddr

Syntax        : ret.l=DBlistaddr(db#)

Description   : This returns the address of the head of the Nodelist
                which can then be passed to functions which require a
                standard Amiga namelist as a parameter.

Example       : ; display our list of names in a GTlistview Gadget
                ; nb. to use this example my GTLIB mod is required.
                GTChangeListM 1,2,DBlistaddr(1)

                            -----------------

Command       : DBfirst

Syntax        : ret.b = DBfirst(DB#)
                        DBfirst DB#

Description   : This command sets the current record pointer to the
                first record in the database, if the database is empty
                or undefined the function will return 0, otherwise it
                will return 1.

Example       : ; Set pointer to first record in our database
                ok=DBfirst(1)

                            -----------------

Command       : DBlast

Syntax        : ret.b = DBlast(DB#)
                        DBlast DB#
```

```
Description     : This command sets the current record pointer to the
                  last record in the database, if the database is empty
                  or undefined the function will return 0, otherwise it
                  will return 1.

Example         : ; Set pointer to last record in our database
                  ok=DBlast(1)

                            -----------------

Command         : DBnext

Syntax          : ret.b = DBnext(DB#)
                          DBnext DB#

Description     : This command sets the current record pointer to the
                  next record in the database, if the database is empty,
                  undefined or there are no more records the function will
                  return 0, otherwise it will return 1.

Example         : ; Scan our database records, start to finish
                  ok=DBfirst(1)
                  while (ok)
                    ok=DBnext(1)
                  wend

                            -----------------

Command         : DBprev

Syntax          : ret.b = DBprev(DB#)
                          DBprev DB#

Description     : This command sets the current record pointer to the
                  previous record in the database, if the database is empty,
                  undefined or there are no more records the function will
                  return 0, otherwise it will return 1.

Example         : ; Scan our database records, finish to start
                  ok=DBlast(1)
                  while (ok)
                    ok=DBprev(1)
                  wend

                            -----------------
Command         : DBadd

Syntax          : ret.b = DBadd(DB#,recvar)
                          DBadd DB#,recvar

Description     : This adds the values stored in the record variable to the
                  database at the current position. If it cannot be added, the
                  function will return 0. If it adds OK, 1 will be returned.
                  (In addition, if the add had to expand the size of the
                  database, this function will return a 2, this is for
                  information Only).
```

```
                  If the database is keyed, the data is added at the
                  correct position, to keep the database in order.

Example         : ; Add a record to our database
                  StrToFls "Joe Bloggs",test\name,30
                  StrToFls "Joes House",test\addrs,60
                  test\age=32
                  ok=DBadd(1,test)
                  If ok then Nprint "Yippee, added a record"

                           -----------------


Command         : DBaddLast

Syntax          : ret.b = DBaddLast(DB#,recvar)
                          DBaddLast DB#,recvar

Description     : This adds the values stored in the record variable to
                  the end of the database. If it cannot be added, the
                  function will return 0. If it adds OK, 1 will be returned.
                  (In addition, if the add had to expand the size of the
                  database, this function will return a 2, this is for
                  information Only).
                  If the database is keyed, the data is added at the
                  correct position rather than at the end.

Example         : ; Add a record to our database
                  StrToFls "Joe Bloggs",test\name,30
                  StrToFls "Joes House",test\addrs,60
                  test\age=32
                  ok=DBaddLast(1,test)
                  If ok then Nprint "Yippee, added a record"

                           -----------------

Command         : DBaddFirst

Syntax          : ret.b = DBaddFirst(DB#,recvar)
                          DBaddFirst DB#,recvar

Description     : This adds the values stored in the record variable to
                  the start of the database. If it cannot be added, the
                  function will return 0. If it adds OK, 1 will be returned.
                  (In addition, if the add had to expand the size of the
                  database, this function will return a 2, this is for
                  information Only).
                  If the database is keyed, the data is added at the
                  correct position rather than at the start.

Example         : ; Add a record to our database
                  StrToFls "Joe Bloggs",test\name,30
                  StrToFls "Joes House",test\addrs,60
                  test\age=32
                  ok=DBaddFirst(1,test)
                  If ok then Nprint "Yippee, added a record"
```

Function : DBrecs

Syntax : ret.l=DBrecs(DB#)

Description : Returns how many records are stored in the database.

Example : Nprint "Database has ",DBrecs(1)," records in it"

Command : DBget

Syntax : ret.b=DBget(DB#,recvar)
DBget DB#,recvar

Description : Retrieve the current record from the database into the
record variable. If ok, the function returns 1, if the database is empty or undefined 0 is returned

Example :
```
; lets get some data
ok=DBfirst(1)
if ok
  DBget 1,test
  Nprint "Name   :",FlsToStr$(test\name,30)
  Nprint "Address:",FlsToStr$(test\addrs,60)
  Nprint "Age    :",test\age
end if
```

Statement : DBkill

Syntax : DBkill DB#

Description : Remove the current database from memory, if you do not remove a database it will be removed automatically when the program finishes.

Example :
```
; I don't want ya no more, o database of mine
DBkill 1
```

Statement : DBdelete

Syntax : DBdelete DB#

Description : Delete the current record from the database.
NB. To keep the speed of the library at a maximum, deleted records are NOT reallocated. Therefore if you do a large number of deletes it may be worth reorganising the database. This can be performed by saving it off (eg. to ram:) and reloading it.

```
Example         : ; I hate that first record
                  ok=DBfirst(1)
                  if ok then DBdelete 1

                            -----------------

Command         : DBsetpos

Syntax          : ret.b=DBsetpos(DB#,record#)
                        DBsetpos DB#,record#

Description     : Positions the record pointer at record#, if record#
                  is greater than the number of records in the database
                  it will make the last record current.

Example         : ; I wanna be, at record number 3
                  DBsetpos 1,3

                            -----------------

Statement       : DBcasesense

Syntax          : DBcasesense ON|OFF

Description     : Switch case sensitivity on or off for database searches
                  and adds to keyed databases.

                             -----------

Statement       : DBsetkey

Syntax          : DBsetkey ON|OFF

Description     : Switch keying on or off for database additions.
                  NB. If you switch off case sensitivity then add a record
                  to a keyed database the database may no longer be in order,
                  as yet there is no sort command to reverse this situation.
                  Additions to an unkeyed database are MUCH faster.

                             -----------

Function        : DBmemtype

Syntax          : DBmemtype memtyp

Description     : Set type of memory to be used when creating new databases.
                  FASTRAM = 0
                  CHIPMEM = 2
                  CLRMEM  = 65536

                             -----------
Function        : DBfind

Syntax          : ret.b=DBfind(DB#,search$[,length,offset[,startrec]])

Description     : Search database from the beginning for a string.
```

```
                  if length and offset are not supplied, the whole
                  record is searched. If a record is found, 1 is returned
                  and the record is made current. 0 is returned if the
                  search fails. If you only want to search part of the
                  record, use the offset to indicate how many bytes from
                  the start of the record you want to start, and set length
                  to the number of bytes to search.
                  If startrec is supplied the search will start from the
                  indicated record.

Example         : ; Find joes house by searching address fields
                  ok=DBfind(1,"Joe",60,30)
                  if ok
                    DBget 1,test
                    NPrint "Yeehaaa, joes still here"
                    Nprint "Name   :",FlsToStr$(test\name,30)
                    Nprint "Address:",FlsToStr$(test\addrs,60)
                    Nprint "Age    :",test\age
                  end if

                             -----------

Function        : DBfindnext

Syntax          : ret.b=DBfindnext(DB#)

Description     : Search for the next occurance of search$ in the database.
                  If a record is found, 1 is returned and the record is made
                  current. 0 is returned if the search fails.

Example         : ; Find all joes houses
                  ok=DBfind(1,"Joe",60,30)
                  while (ok)
                    DBget 1,test
                    NPrint "Yeehaaa, joes still here"
                    Nprint "Name   :",FlsToStr$(test\name,30)
                    Nprint "Address:",FlsToStr$(test\addrs,60)
                    Nprint "Age    :",test\age
                    ok=DBfindnext(1)
                  wend

                             -----------

Statement       : DBupdate

Syntax          : DBupdate DB#,recvar

Description     : Updates the current record with the data held in recvar.
                  If the database is keyed, it will be reinserted at the
                  correct position.

Example         : ; Let Jim have Joes House
                  DBget 1,test
                  StrToFls "Jimmy Jones",test\name,30
                  DBupdate 1,test

                             -----------------
```

```
Command         : DBload

Syntax          : ret.b=DBload(DB#,filename$)
                        DBload DB#,filename$

Description     : Load a database from disk. If the database is already
                  in use it will be destroyed. If the load fails the
                  function will return 0, if OK it will return 1.

                            -----------------

Command         : DBsave

Syntax          : ret.b=DBsave(DB#,filename$)
                        DBsave DB#,filename$

Description     : Save a database to disk. The database is reorganized
                  as it is saved, removing any deleted records. If the
                  save is OK, 1 will be returned, if it fails 0 will be
                  returned.

                            -----------------

Function        : DBisnext

Syntax          : ret.b=DBisnext(DB#)

Description     : Tells you if there is a next record in the database.
                  See, the example program for possible uses.

                            -----------------

Function        : DBisprev

Syntax          : ret.b=DBisprev(DB#)

Description     : Tells you if there is a previous record in the database.
                  See, the example program for possible uses.

                            -----------------

Function        : DBcurrent

Syntax          : ret.l=DBcurrent(DB#)

Description     : Returns the current record number (0=database empty or
                  not defined)

                            -----------------

Function        : DBmodified

Syntax          : ret.l=DBmodified(DB#)

Description     : Returns TRUE if the database has been modified since
                  it was loaded or created.
```

Function : DBactive

Syntax : ret.b=DBactive(DB#)

Description : Returns True if the database is active (ie. defined)
returns false otherwise

Statement : DBpush

Syntax : DBpush

Description : stores the current database pointer position

Statement : DBpop

Syntax : DBpop

Description : sets database pointer to the last record stored by DBpush

---------- End Of Library Documentation ----------