KEY_TO_TRITON

Philipp Lonke <phips@scout.franken.de>

COLLABORATORS

	TITLE : KEY_TO_TRITON	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Philipp Lonke <phips@scout.franken.de>	June 24, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

KEY_TO_TRITON

1.1 The key to TRITON programming in Blitz2

```
                        TRITON step by step
                        -------------------

   A small key to TRITON programming using the interfaces written
              by Philipp Lonke <phips@scout.franken.de>


Introduction              What's all about.
Needful things            What should be known before
Installation              How to get started
Programming example       "Hello world" in TRITON
FAQ                       Frequently asked questions
Addings                   What's more to say?
The future                What could come...
Thanks and legals...      The end!
For advanced               How to create smaller execs

Changes                   What changed since the first release?

About the author          How to contact the famous
                           guy who programmed this
                           conversion :-)
Blizzing                  Some things for Blitz2
```

1.2 Introduction

```
First of all, forget everything about GUI programming in Blitz2. It
is not the same creating a GadTool-GUI or a TRITON GUI.

Read this documentation carefully, so you really understand the
difference.

          Go back
```

1.3 Some useful definitions

```
When speaking of TRITON, there are to major terms:

    an application: That is your program. The informations
                    (application tags) are for use in the
                    TRITON Preferences Editor (ShareWare)

    a project     : In fact, that's your GUI. For every
                    window you use, you create a new project.

Second, every string passed to any TRITON library function has
to be passed using Null(s$) because these functions need the
adress of a null-terminated string.

*NEVER* use an ID of zero for anything (window, button etc.)

If you ever break your program with the debugger when the window
was already open, do *NOT* end it with the debugger, instead
continue the program and close the window. Otherwise TRITON will
not close the window what causes confusion when you restart your
program!

         Go back
```

1.4 How to install TRITON on your Blitz2 system

```
First of all: Before you can use this package, you have to get at least the
TRITON developer archive from aminet/dev/gui. Take also a look at
the TRITON user archive in aminet/util/libs. It contains the very good
TRITON preference editor.

IF YOU DON'T HAVE THESE ARCHIVES, THIS PACKAGE IS WORTHLESS TO YOU!

Since this release (2.2), the triton.library is precompiled for Blitz.
So - in fact - you only need the Triton user archive but the autodoc
files are only avaiable in the developer archive!

triton.library1 and TagListLib.obj have their own IDs (given by Leading Edge).
So you mustn't care about converting them.

You only have to check if ID 219 (for triton), ID 26 (for TagListLib)
and ID 48 (for DBaseLib) are free on your system. This should be no problem
 using LibMan or ViewLibs.

Now simply copy tritonblitz/blitzlibs/amigalibs/triton.library1 into
BlitzLibs:amigalibs/

and

tritonblitz/userlibs/TagListLib.obj and DBaseLib.obj into your
BlitzLibs:userlibs/ directory.

Now delete your old "deflibs" file and create a new one with
```

```
MakeDefLibs, or, if you use Blitz2.1, run LibMan.

Reload Blitz2 and try typing "TR_OpenProject_" (without quotes) and
press the HELP-Key on your keyboard. Do the same typing
"InitTagList" (without quotes) and press the help key again. Try typing
"StrToFls", this should change to token colour, too.

If it turns to your tokencolor and a short helptext appears in your titlebar,
then you did everything right.

To be able to use the TRITON include-file "triton.bb2", you have
to save it first with Blitz2, so that it will be tokenised!
```

1.5 How to program a TRITON user interface?

```
See here for library details

The use and syntax of the commands of TRITON.library can be taken
from the autodoc file which comes with the TRITON developer archive.

Remember that you are using system library calls. You get all the
keywords from the TRITON archive's autodoc file, just add an under-
score to them. You mustn't write i.e. TR_OpenProject but instead
TR_OpenProject_

If you plan programs with more than 5 taglists, don't forget to
increase the amount in 'compiler options' !


Now we can start:

Of course, first come the includes (remember amigalibs.res!) and
the most important variable initialisation.

    INCLUDE "blitz2:bbincludes/libraries/TRITON.bb2"    ; or wherever you
                                                        ; put it.
    InitTaglist 2,10                    ; making place for our application
    InitTagList 1,20                    ; making place for 20 tags
                                        ; in the first taglist

    Use TagList 2                       ; and use the app. taglist

Now, we could tell TRITON, that we launch a new application, our
program. Therefore, TRITON wants to know something about our
program for its prefs program.

The tags you can use, are:

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

And it looks like this in your code:

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

Now we open our application. Remember that this variable MUST be long!

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

Before we continue, we have to check that nothing has happened.

```
    if (application)

        ... code ...
```

Now we create our GUI. First, we want to do a window with two buttons. So we use the prepared Taglist:

```
    Use TagList 1
```

Therefore we use the macros. For the window, the most important ones are:

```
    !WindowID{id}   - the ID must not be zero!!!

    !WindowPosition{positionflag}

    !WindowTitle{Null("Window title")}
        as you remember, strings and text must always
        be passed with the Null(string) command!!

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

If you create your Taglist with the TagListLib, then remember that the last Tag must be #TAG_END,0

Now we come to our Buttons. All Gadget are grouped in TRITON. There are two major kind of Groups: horizontal and vertical aligned groups. Depending on if you want your gadgets horizontal or vertikal aligned :) you must choose between these groups. Of course, they can be mixed. So you can make 4 Buttons in 2 horizontal groups and put these groups into a vertical group. Every group must be ended with the !EndGroup macro.

```
    Groups : !HorizGroup, !VertGroup
                (arrangement of buttons: look into the include file!
                 A Group must always end with the macro !EndGroup)

    Buttons: !Button{Null("T_ext"),id}
```

```
                (Shortcuts are marked by an underscore in front of the Key ↩
                   )
```

The code looks like this:

```
    AddTags !VertGroupA
    AddTags     !Space
    AddTags     !HorizGroupA
    AddTags         !Space
    AddTags         !Button{Null("_Save"),12}
    AddTags         !Button{Null("_Cancel"),15}
    AddTags         !Space
    AddTags     !EndGroup
    AddTags     !Space
    AddTags !EndGroup
    AddTags #TAG_END,0
```

You should type your code always structured to keep the context
in mind. It's easier to overview ;)

Now we can open our window, i.e. our project which MUST also be
a variable of long!

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

Now comes the real program:

```
    if (project)   ; only if no error occured

        close_me.b=False      ; let the window open

        while NOT close_me    ; and as long as it's open
            TR_Wait(application,0)     ;wait for a message

            *trmsg.TR_Message=TR_GetMsg_(application)
                           ; what does the user do??

            while (*trmsg)   ; as long as it's valid

                if (*trmsg\trm_Project=project)    ; it's for our window

                  select *trmsg\trm_Class       ; which message?

                    case #TRMS_CLOSEWINDOW
                        close_me=True

                    case #TRMS_ACTION           ; a button was triggered

                      select *trmsg\trm_ID      ; which one?

                        case 1
                            ; button 1
                            ; code

                        case 2
```

```
                                ; button 2
                                ; code

                            end select

                           case #TRMS_NEWVALUE     ; check for i.e. checkboxes
                                                   ; and some other gadgets which
                                                   ; return this message instead
                                                   ; of #TRMS_ACTION. You need it ↩
                                                       only if
                                                   ; you have such a gad in your  ↩
                                                      GUI.
                      end select
                    endif

                    TR_ReplyMsg_ *trmsg     ; always reply to a msg as
                                            ; fast as possible!

                    *trmsg=TR_GetMsg_ (application) ; and get the next

                wend
              wend

          TR_CloseProject_ project          ; close our window
        else
            NPrint "Unable to create project"       ; if it failed
        endif

        TR_DeleteApp_ application           ; and tell TRITON that
                                             ; our program was terminated
    else

        nprint "unable to create application"   ; if it failed

    endif

      Free Taglist 1              ; give the taglists memory free

    end
```

To see all the macros and constants, take a look at the Blitz
include file "TRITON.bb2"or at the C include file "triton.h"

To see how to do other gadgets (some send TRMS_NewValue messages!) and what
more functions the TRITON library offers, take a look at the autodoc file,
the demolistings or ask me directly.

The use of QuickHelp is shown in TOOLMANAGER1a.bb2, also the
use of Blitz2-Lists for the ListView

A special case is TRITON s Easyrequester, which
replaces the system’s requester.

Go back

1.6 TRITON ListViews

You can use Blitz2-Lists for TRITON's Listviews which makes them easy to use.

Here the source taken out from {i}TOOLMANAGER1.bb2

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

NEWTYPE .LVItem

  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}       ; important!! &LVNodes(0) ↩
   -36
AddTags     !EndGroup
```

A smarter way to create a Listview-List is to use the DBaseLib by G. Kennedy. It comes with many commands for a very easy list handling and is more system compliant.

Here's the code, taken from 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

possible Flags are:

#TRWP_DEFAULT

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

Go back
```

1.8 Windowflags

```
      the flags are combined with "OR" or "|". Possible
      flags are:

      #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

Go back
```

1.9 TRITON's easyrequester

```
The function TR_EasyRequest_ (app,body,gads,tags) creates a requester
as the Blitz-Function Request does.

body : Null("This is the requester text")
gads : Null("Ok|Try again|Quit")
tags : you can use the following tags:

       #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: How can I change the contents of a listview?

A: That's very simple. Do it this way:

    TR_SetAttribute_ project,id,0,NOT 0

    ; code to change list contents

    TR_SetAttribute_ project,id,0,&List(0)-36   ; if you're using
                                                ; Blitz-Lists

    ; or, with DBaseLib
    TR_SetAttribute_ project,id,0,dblistaddr(listnr)

 Q: I have a ReturnOK-Button and a string gadget in my window.
    Everytime something is entered into the string gad and con-
    firmed by return, the button is triggered. How can I avoid
    this?

 A: Use the macro !StringGadgetNR instead of !_StringGadget.

 Q: I want to use a fixed width font, how to do it?

 A: Very easy: When you do your windowtags, just add this line:

    AddTags #TRWI_FixedWidthFontAttr,font

    where font is initialised as font.TextAttr=Null("name.font"),size

            If you just want to use the system's fixed width font, then you
            can go on and use the !FW... macros without setting the
            #TRWI_Fixed... constant.

 Q: How do I get the string of a string-gadget?

 A: You have to peek$ to the pointer.

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

 Q: When I use the macro !ListROC{}, I always get an error
    message!

 A: You mustn't use the macro after a AddTags command, because the
    macro itself contains this command.

    ; example
    AddTags !VertGroupAC
```

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


     Go back
```

1.11 Some final words

I think you got now the difference between a GadTools (or, worse, a Blitz) GUI and TRITON. But from now on you just don't need to care about fontsensitivity and calculating positions of gadgets.

You should only take the above example as a model to program a TRITON GUI. I have to admit that I didn't change the other demo listings to this way, so just take them to see how to create other gadgets or layouts. Take care of these rules:

a) use goto and/or gosub rarely in your program. This is a not very good style which should be avoided as often as possible. Blitz2 offers many possibilities for it: statements and functions.

b) Every macro that has the same name as a Blitz2-Keyword begins with an Underscore. So the original macro !StringGadget{...} is named !_StringGadget{...}. If a TRITON macro turns yellow in your TED, just put a "_" in front :)

c) Before getting a message, use TR_Wait_ app,otherbits

d) Always check that your project/application was opened!

e) Reply every Message you get from TRITON.

f) To program, use the macros - they are the easiest way to create a TRITON GUI. To explain all macros would exceed this little docu, but just take a look at either the Blitz2-Includes or the original C-Includes (in "TRITON/developer/includes/libraries"). The name of all constants and macros should be self-explaining. Try them out!

   You really should take a deep look at the Blitz2-include file TRITON.bb2, just to see what kinds of gadgets you can create and what the macro names are for!
   And you should also take a even deeper look at the autodoc file in TRITONs developer archive.

But always remember: Due to the TRITON Preferences Editor, the user can not only change the look of the gadgets but also place your window(s) on any screen he likes. So do never use fixed coords! In fact, you do not need to size your window - the user can change it and it will be safed in ENV: and ENVARC:, so with

every startup of your program, the window opens in the same
dimensions and coords where the user closed it last.

About this Guidefile

You should keep in mind, that this file does some system calls to show you the include file and some sources. If you change the path or the location of this guidefile, these calls may end in an error.

Go back
Authors adress

1.12 Contact the author

If you have any suggestions, ideas or if you just need a little help, contact me

via eMail: phips@scout.franken.de

in the BlitzBasic Mailing list

via SnailMail: please understand that you can't reach me by SnailMail. Use eMail instead.

If you really don't reach me, just write to the programmer of TRITON, Stefan Zeiger (adress in the orig. docu!), he surely knows where I am and how you reach me - he's nearly a neighbor of mine :)

Go back

1.13 The end and the future

So I wish you happy blitzing with TRITON and hope to see some of your programs which use TRITON. I'd really appreciate, if you send me just a few words when you finished a program that uses TRITON, so we are able to create a TRITON-Applications-list as exists for MUI.

I included a little program called "memo" to this package - it can only be started via CLI and pops up a TRITON requester with your text in it. You need two arguments!

memo "Hello World!" "Remember to write Philipp!!"

This program could be used with CyberCron, DCron etc. for reminding.

Go back

1.14 Thanks and more go to...

```
Thanks go to these people (in no order):

    · Rupert Henson, who helped me very much creating the macros
    · D.C.J. Pink for his TagListLib
    ·~Patrik Rådman (pradman@mail.abo.fi) for TaglistLib 1.1
    · of course to Stefan Zeiger for TRITON
    · and to ACID Software for BlitzBasic2
    ·~Irena, my girlfriend for everything
    · Michael Bergmann for nice calls, interesting discussions
      about computers in any (im-)possible corner of the world
    ·~James Savage for his trying to get TRITON to work :-)
    · Graham Kennedy for some nice advices for TRITON and
      for his DBaseLib
    · Stefan Haefner for its insisting on a german guide file
    ·~and all the other I forgot...

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

TagListLib.bb2 is freeware. Triton is shareware. DBaseLib is freeware

Legal stuff: THIS PACKAGE IS PUBLIC DOMAIN.I TAKE NO GUARANTEE FOR
    ANYTHING THAT HAPPENS TO YOU AND/OR YOUR MACHINE BY USING IT.
    BUT IF YOU CHANGE THE CODE PLEASE SEND ME A COPY OF IT SO I
    CAN ALWAYS BE ABLE TO UPDATE IT IN FUTURE RELEASES!


keep on blizzing,
        phips@scout.franken.de


         Go back
```

1.15 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.

Go back

1.16 About BlitzBasic2

BlitzBasic 2.1 is (c) by Acid Software
LibMan is (c) BlitzBasic Distribution Köln and
written by Peter Eisenlohr

Subscribe to the BlitzBasic-Mailing-list, if you like:

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

You'll get a mail telling you how to subscribe.

BlitzBasic-FTP sites are:

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

1.17 Changes since release 1.0

I eliminated some typos. If you still find some, write me immediately!

```
I tried to fix the macro !ListROC{} but couldn't finde the bug. Blitz
still reports the macro being to long. Sorry that you can't use it, you
just have to do it "by hand".

Release 1.1

Eliminating typos (as always :-)
The macro !ListROC{} doesn't work.

2.0

Could change the macro !ListROC with the Taglistlib so it
could work now.

2.1

german translation of the guide file.
This release was included in german BUM 9.

2.2

Adding DBaselib to the archive.
All libraries (triton,Taglist,Dbase) got precompiled and
fix IDs by RWE.
```

1.18 For advanced programmers

```
There is a way to make your executables a lot smaller when using
TRITON.

First, you don't have to use Null() to pass strings to TRITON. You
could type &string$ instead as the Blitz2-Strings are all
zero-terminated. I recommend Blitz2 v1.9 at least to be sure that
this will work!
Remember then, that the strings must not be changed until they have
been used by TRITON, which is usually after TR_CreateApp_

You can also use a label reference to pass the strings. Remember,
that if you use Functions/Statements, you have to create new
labels for each Function/Statement!.

To get a view how these tricks work in reality, read the source
of the listing "memo2.bb2" which is written and commented by
Daniel Pink.
```

1.19 Library details

```
TagListLib documentation
DBaseLib documentation
```

1.20 TagListLibrary

```
                TagListLib for Blitz Basic 2 - short documentation.

TagListLib was originally written by D. Pink for the Triton Blitz conversion
by Ph. Lonke. This version 1.1 was bugfixed & enhanced by me.

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

Here's a short list of the commands:

o InitTagList TagList.w,NoTags.l
  * Allocates memory for a taglist
o AddTags     [TagList.w] [[,Tag.l,Data.l]]
  * Tag,Data can be repeated, ie AddTags #TAG_1,100,#TAG_2,200,...
o NoTagsLeft  [TagList.w]
  * Returns number of tags left in Taglist
o TagList     [TagList.w]
  * Returns location in memory of Taglist

(For a practical example, take a look at TagListLib_example.bb2)
```

1.21 DBaseLibrary

```
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
```