TRUE BASIC

The Original BASIC

FORMS

Visual Interface Builder

FORMS Visual Interface Builder

for True BASIC 6

Introduction

True BASIC contains three built-in sub-routines called:
OBJECT
SYS_EVENT
TBD (and TBDX)

These routines are very complex and quite difficult to use. As usual, True BASIC has attempted to organize what these sub-routines do into a logical and coherent set of easy to use routines. These routines have been gathered together into two library modules called TrueCTRL and TrueDIAL.

The first of these library modules allows the user to create windows and to create controls and objects to put inside them. The difference between "controls" and "objects" is that controls such as push buttons, check boxes and lists respond to mouse clicks, whereas objects such as text, images, lines and circles don't respond to mouse clicks. Although the way these routines have been organized is very good, it is still a lot to remember as you can see from chapter 14 and chapter 22 in the True BASIC Users Manual. Moreover, there is little control over color and the choice of fonts with these objects, because they are supplied by Windows.

A new library called CTX is an attempt to rectify the shortcomings of the TrueCTRL library by giving the user a greater range of objects, with complete control over color and fonts. This is possible because the objects are not produced by Windows; they are individually drawn with True BASIC graphics. The list of additional objects includes a right click floating menu, and a multi-column list object.

The second library module deals exclusively with dialog boxes, i.e. a small window that pops up containing a message or an instruction and often has some push buttons giving the user a choice. The program halts while the dialog box is on screen and the only way to continue is to make a choice by clicking one of the buttons. This is a very flexible way of obtaining user input. There are only ten dialog boxes including get-file and save-file boxes, so it is not too difficult to remember how to use these. Again there is the problem of lack of control over the color and the font.

Again, a new library called TDX is an attempt to address the shortcomings of the TrueDIAL library. The range of dialog boxes has been increased and there is complete control over color and fonts. Once again the dialog boxes are created entirely from True BASIC graphics. The list of additional dialog boxes includes a color selector and a font selector.

The program called TBF.EXE is an attempt to address the problem of complexity. Essentially TBF.EXE creates windows automatically and then allows the user to pick objects from a toolbar and drop them anywhere in the window. For convenience we use the word FORM to describe a window that is generated automatically. A rubber box technique also allows the user to move the object around and to re-size the object. A “properties” box allows the user to customize the object in terms of color, fonts and various other attributes.

The program TBF.EXE automatically appears under the HELP for True BASIC menu in the True BASIC editor as the item FORMS. You can dip into TBF.EXE directly from the program you are currently developing in order to edit existing windows or to create more. When you return to your program it will contain all the necessary code to reproduce the window and its objects that you have just designed. True BASIC allows up to ten windows and TBF.EXE has the same limitation.

Now a few words about SYS_EVENT. In the good old days of DOS the programmer remained in control all of the time. A program would start at the beginning and proceed via numerous branches in logic to the end. Periodically the program might require some input from the user but this was only a pause in the path of the program towards the end. Windows programming is completely different. In this case the programmer has absolutely no control over what the user might do next. Who can tell which object the user might click on? The way round this problem is to set up a DO…LOOP and then scan the keyboard and the mouse for any sign of input. This is what SYS_EVENT does. As usual it is a little too complicated to use so TrueCTRL contains a simpler version called TC_event. Once the program has detected an event i.e. the user has used the keyboard or the mouse, then you need a whole bunch of code to sort out which event it is and which object it refers to. Chapter 20 in the True BASIC Manual gives you an idea of how many events there are and what they all mean. Let us assume you have determined what the event is and which object is concerned, you now write a small sub-routine that deals with that event before returning to the DO….LOOP. All Windows programs work this way. The program structure is very simple and always the same. True BASIC even supplies a “skeleton” program that you can use as a model for all your programs.

How does the program know where all the objects are? The size and location of all windows are defined by four co-ordinates in this order; left, right, bottom, top relative to the whole screen. The position of objects inside windows uses the same system of co-ordinates but they are relative to the window itself.

There are basically two units of measurement; USER units and PIXEL units. At present TBF.EXE works in PIXEL units. Later versions may allow you to use USER units. USER units run from 0 to 1 left to right across the screen and from 0 to 1 up the screen, whereas PIXEL units run left to right, but top to bottom.

The great advantage of user units is that you don’t have to worry about how big the screen is in terms of pixels, so your program will run and look the same whatever the screen definition. The disadvantage is that location co-ordinates for windows less than full screen size are always in decimals and often to 3 places. It is also very difficult to relate user units to point sizes when you are defining fonts.

The advantage of using pixel units is that location co-ordinates are always in whole numbers, and it is easier to work with different font sizes because the point size is directly related to pixels. The big disadvantage is that pixel units are dependent on the screen size. Not that you need to worry about co-ordinates or units because TBF.EXE works out the co-ordinates for all the windows and objects you create.

The final concept that you need to get your head around is that all windows and all objects inside windows each have a unique identity or reference number. How else would we know which object had been clicked with the mouse. Strange as it may seem, at no time do you need to know the value of the variable name. Once again, you don't need to bother with this because TBF.EXE supplies a default variable name for each object, but the properties box allows you to change this to a variable name of your choice. For your own convenience you are strongly advised to use your own variable names instead of the default names.

The CTX library also has a similar routine for dealing with events called CT_event.

As a general rule of thumb, all TrueCTRL routines begin with TC_ and all CTX routines begin with CT_. The routine names in both libraries are exactly the same apart from the prefix. Moreover, you can use routines from both libraries in the same program. In the case of the CTX library the number of different types of event has been reduced to only four, which is not too hard to remember.

TBF.EXE has a built-in skeleton program that is more extensive than the standard skeleton. It deals with events itself, including MENU events, and diverts the program flow to ready made sub-routines waiting for you to add you custom code. The sub-routines even have a reminder at the top telling you what the sub-routine is supposed to do. For example, let us suppose your window has three push buttons labelled push1, push2 and push3. At the end of your automatic program there will be three sub-routines with names like:

```
SUB action_push3
REM: what do you want to happen when the push button push3 is clicked.
END SUB
```

In some cases there will also be a useful line of code automatically generated. For example, if the object is a list then you will want to know which item in the list has been chosen by the user. In this case the following code line will provide the answer:

```
   CALL CT_List_Get(lister,selection)      ! list item selected by the user
```

The automatic skeleton is arranged so that all the code relevant to a particular window (FORM) is contained as a block. This block can be edited, added to or even deleted, without affecting any other block. It also makes debugging easier because an error in any window is confined to that block of code.

The skeleton also makes all the necessary library calls and has a built-in error handling routine that sends any error data to a text file called ERRLOG.TXT. Quite apart from the ease of use, the skeleton is an example of good programming structure, that will help you organize the way you tackle your programming tasks.

TBF.EXE does not require you to know anything about TrueCTRL or CTX. You just need to know how to pick objects from a toolbar and how to drag them around the screen to the right position. You don't have to worry about aligning objects because there is a built-in grid that will snap your object to the nearest intersection. A built-in help system shows you how to customize objects with the properties box. You don't have to know about event handling and you don't have to know about units or co-ordinates. You don't have to worry about your program crashing because the error handling system will take care of it. You only need a basic knowledge of the True BASIC language so that you can complete the custom sub-routines.

J.R.Arscott
04 December 2010

Installation

***Note**: The FORMS program is pre-installed in the GOLD edition and can be run directly from the TB Editor program. The instructions below are for installing FORMS in other editions.*

The FORMS program is supplied as a bound program called TBF.EXE. The remaining support files are contained in a zip file called FORMS.ZIP.

When you installed the True BASIC editor, a sub-folder called FORMLIB was also created. This folder contains a program called LINK_FORMS.EXE.

Transfer FORMS.ZIP to this FORMLIB folder and unzip the file.

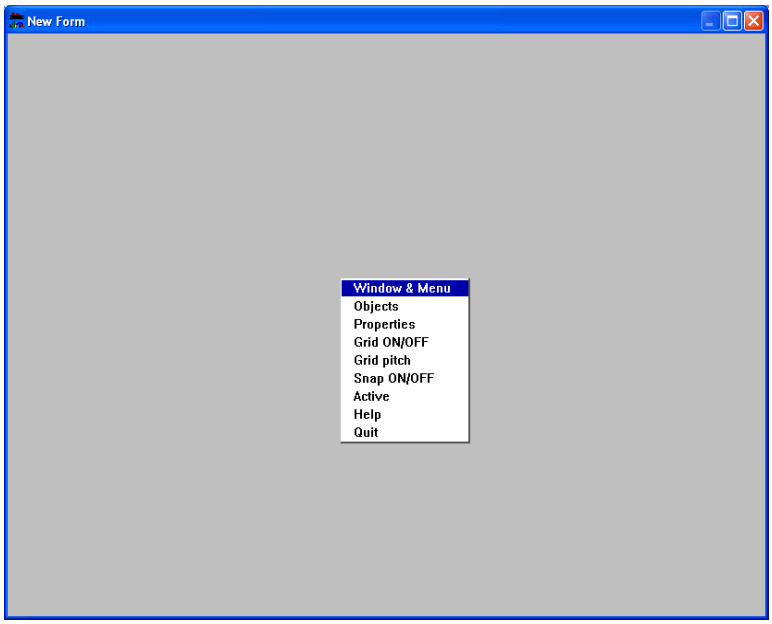
Make sure the True BASIC editor is shut down. Now run the program called LINK_FORMS.EXE. This will enable the FORMS menu item in the True BASIC editor when you next run the editor.

Getting Started

Start up your True BASIC editor in the normal way. Check that the FORMS item under the Help for True BASIC is enabled (it will not be grayed out).

Open a NEW file. Under the HELP for True BASIC menu in the editor select FORMS.

Now the screen will fill with a large gray window and a menu close to the center. The default color is gray (easy on the eyes) and the default title is NEW FORM.



Drag the corners of the blue window frame until the window is the size you want.
Always do this to new windows before adding objects.
Right click to recover the menu then without pressing any buttons, run you mouse over the menu. You will see that each menu item is highlighted as the pointer passes over it.

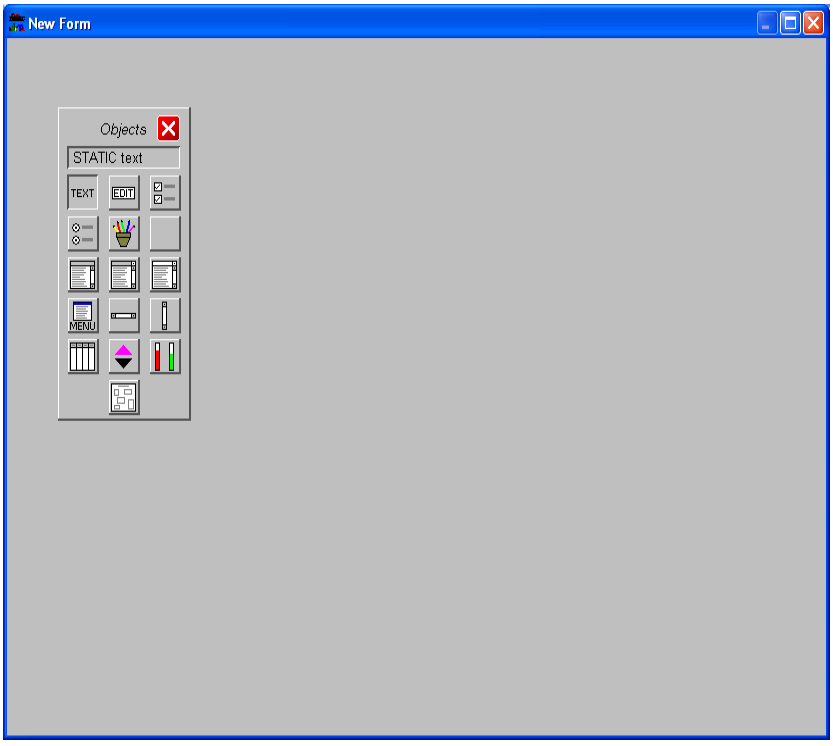
This is the right-click menu. It will be shown whenever you right click with the mouse. The location of the top left hand corner of the menu coincides with the mouse pointer at the moment you click the right button on the mouse.

This is how you always show the right click menu.

Creating Objects

Click on the item labelled OBJECTS. The menu will disappear and the OBJECTS toolbar will be shown on the left hand side of the window.

Click on the first button labelled text.



The button will remain depressed and the title STATIC text will appear above it.

Now click the mouse button anywhere in the middle of the window.

Immediately the words *static text* will appear in black. You have just created your first object. Basically the object consists of a rectangular plate with the words *static text* printed on it. The default background color is the same as the window, which is why you cannot see the plate. Later we can customize this object by changing the text, by changing its color and by changing the font. The default background color is gray, the default ink color is black and the default font is 10-point SYSTEM BOLD.

This is how you create all objects.

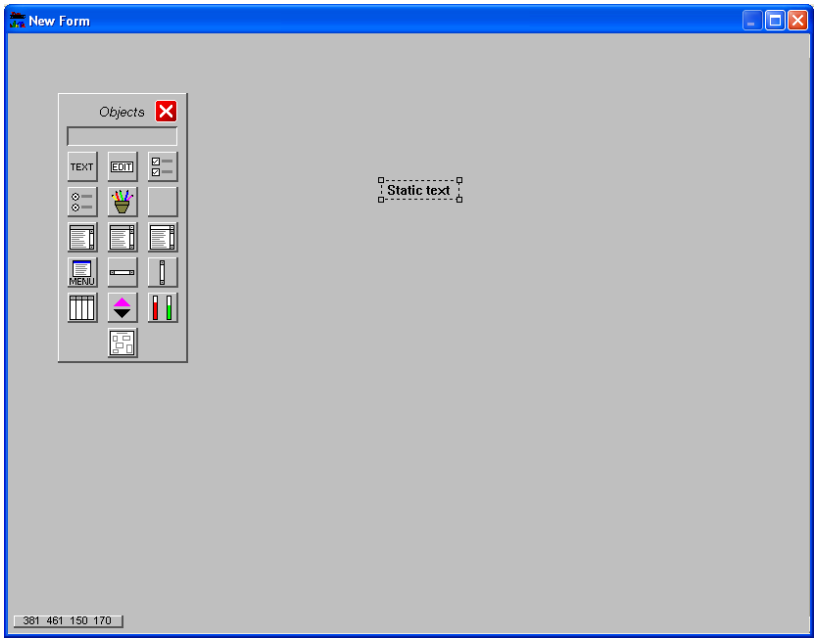
Before we mess about customizing the object we want to make sure it is in the right place. Click on the object and a dotted line will appear around the perimeter. There are four mini boxes in the corners. Point the mouse anywhere inside the dotted box then hold the mouse button down and drag the dotted box somewhere else in the window then release the mouse button.

This is how you move all objects.

Now point the mouse into a corner box and holding the mouse button down, drag the corner to make the dotted box longer and wider. Then release the mouse button. Notice the small numbers in the bottom left of the window show the co-ordinates of the object as you change its size.

This is how you re-size all objects.

To freeze the object in its new location and at its new size just click the mouse anywhere outside the dotted box. The dotted outline will now disappear.



If you make a mistake at any time, click on the object to produce the dotted outline to make the object the focus, the press the DELETE key on the keyboard.

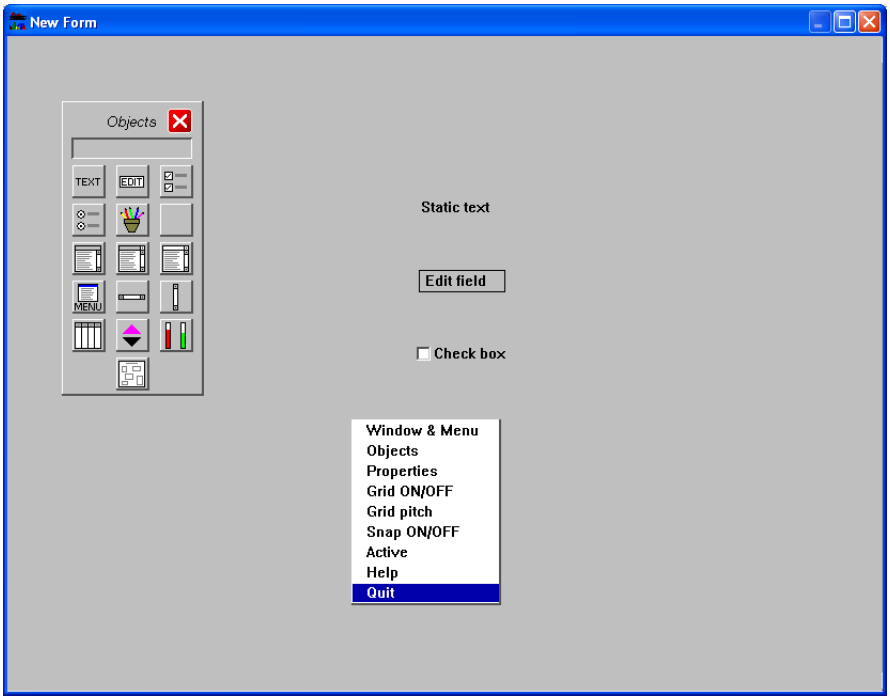
This is how you always delete objects.

Now let us create the next object. Click on the EDIT button in the object toolbar.
Then click the mouse anywhere in the window. Just as before an EDIT object will be created.

Next select the third button in the object toolbar – this is known as a CHECKBOX object. It consists of a small white box and a label alongside. Now click the mouse anywhere in the window. Just as before a CHECKBOX object will be created.

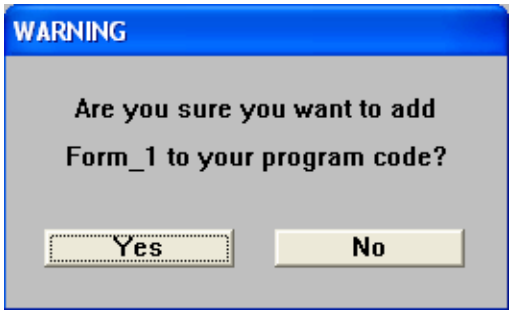
You have successfully created three objects. Now let us see what the automatic code looks like by returning to the TB editor.

First point the mouse at a clear area of the window then use the right mouse button.
The original menu will be displayed. Select QUIT.



This is the way you always return to the TB editor.

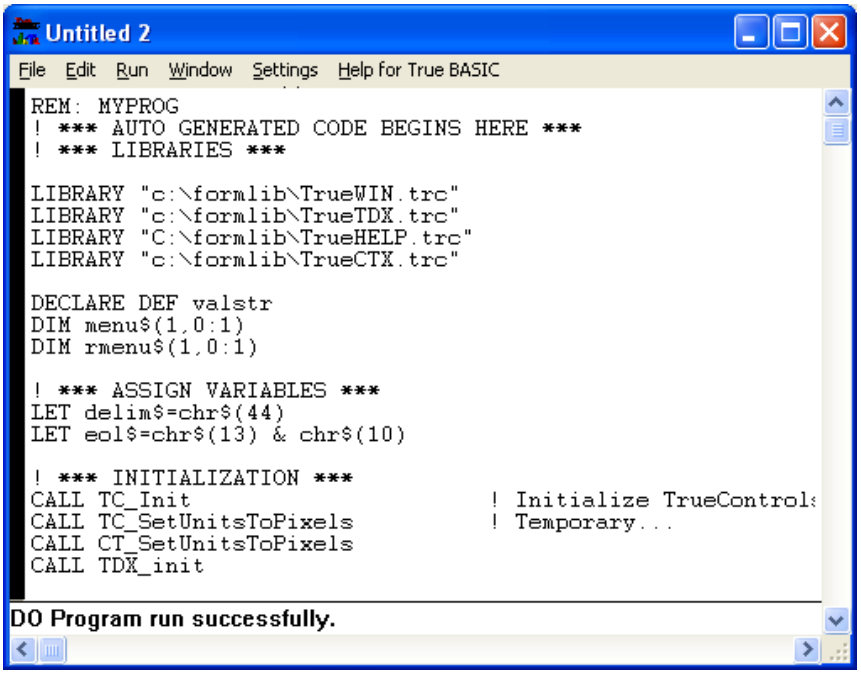
You will be asked if you are sure you want this code added to your program.



If you click the YES button then your program is automatically up-dated with the generated code. If you click the NO button then your original code is returned unchanged.

CAUTION:
If you click the CLOSE window button, this action will shut down the FORMS program, but before you exit the program you will be asked if you wish to save your work, i.e. do you want to add the FORM code to your program.

WARNING:
If you click the CLOSE window button while you are in the middle of selecting, moving or resizing an object then the CLOSE button will NOT work immediately. You MUST complete the action on the object by clicking anywhere inside the window. At that point, the CLOSE window request will be executed.



Look through the code to see what you get for so little effort.

Notice the name you gave the program is reproduced on the first line.

Notice too how the code that refers to FORM_1 is all collected in one block.

At the end of the block you will see three sub-routines.

```
! *** FORM_1 OBJECT ACTIONS (user code) ***
SUB action_id39
   REM: What do you want to happen when the text field id39 is clicked?
END SUB
SUB action_id40
   REM: What do you want to happen when the user completes the edit field id40?
   CALL CT_Edit_GetText(id40,text$)   ! current edit field text
END SUB
SUB action_id41
   REM: What do you want to happen when the checkbox id41 is clicked?
   CALL CT_CheckBox_Get(id41,status)  ! current status
END SUB

! *** END FORM_1 ***
```

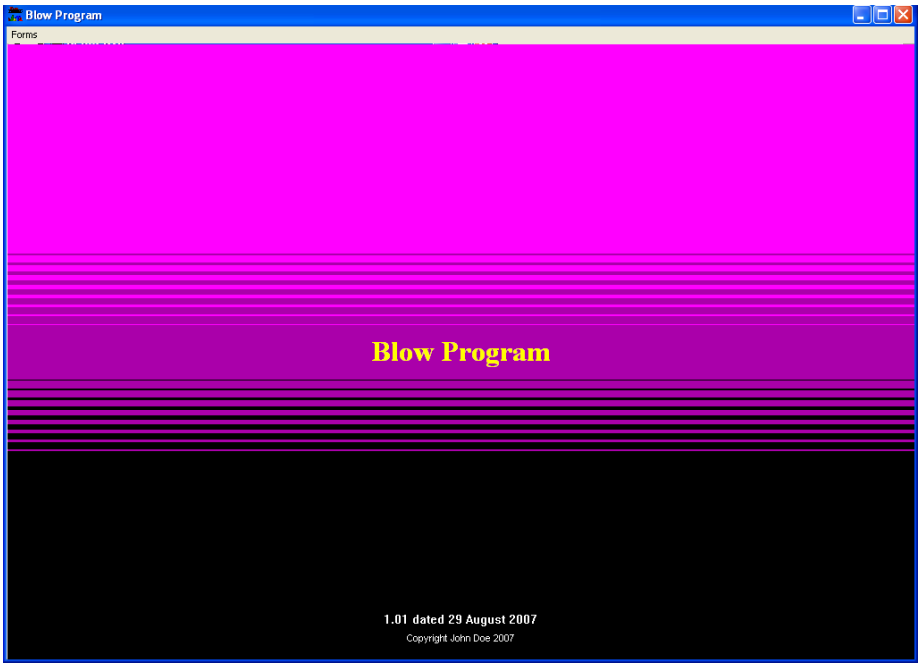
Running Forms

The crucial point to remember is that there is nothing special about FORMS; they are just common everyday windows. In the early days, FORMS were very limited in what you could do with them. They were basically pre-made forms such as invoices, purchase orders, account statements etc. Things have moved on since then but the name lingers on.

Just as it is possible to display several windows on the screen simultaneously, so it is possible to have several forms on the screen at the same time. To help you display your forms the TBF.EXE program uses the menu of WINDOW #0 (the default window) so that you can show as many of your own windows as you like.

Obviously you don't have to use this feature. For example you could use:
CALL TC_show(form1)
To show the first window, and then use a push button with the legend NEXT to show successive windows.

The whole purpose of TBF.EXE is to make things easy for you, which is why the default window provides a means of showing your windows. When you run your program by selecting RUN from the TB editor RUN menu, you will see the default window with the title of your program shown clearly in the center of the screen. Just click on the single menu labelled FORMS and the drop down list will show all the FORMS contained in your program. Click on the form you want and it will be shown.



Notice that the names on the menu are the names on the title bar of each window. As you show each form (window) the task bar at the bottom of your screen will list each window icon. Even if successive windows conceal previous windows, you can just click on the task bar icons to bring that window to the front. Windows can be erased by clicking on the CLOSE button on each window.

A special sub-routine called “SUB splash” has been added to your program that generates the graphic decoration on this “splash” page. If you wish, you can produce your own decoration, or you can display an image.

Note that there is provision for the display of the version number, and provision for the name of the program author in the copyright notice. You can customize these features by changing the appropriate program lines, which you will find at the top of the program.

In this exercise the default window menu only has one item, i.e. the window you just created. The default name for this window is “New Form” so click this item to display your window.

The FORM (window) you have just created will be displayed complete with the three objects.

To return to the TB editor press the ESC key on the keyboard. This is a quick exit that is built into the skeleton program.

You have just created an entire program without typing a line. How easy was that?

In the ACTION sub-routines you must write your own code to deal with each event.

For example, you could add the following code to the first sub-routine

```
SUB action_id39
   REM: What do you want to happen when the text field id39 is clicked?
   SET COLOR 12
   PRINT “I just hit the text object”
END SUB
```

RUN the program again.

Click on the words STATIC TEXT and as you would expect, the words “*I just hit the text object*” will appear in red in the top left of the window.

Note that all CTX objects respond to the mouse, unlike TrueCTRL where static text objects do not respond to the mouse. Having text respond to the mouse can be very useful because you can produce forms that work like HTML. For example, you could have a line of text that says: “Click here to continue”. In TrueCTRL this would be meaningless because nothing happens when you click the text. In CTX clicking a text object produces an event so you will have an “ACTION” sub-routine in which you can put the code that allows the user to continue.

To return to the TB editor press the ESC key on the keyboard.

The way TBF.EXE works is as follows:

The TB editor CHAINS to the main FORM engine called TBF.EXE. The first thing that TBF does is to read a temporary file to fill its own internal array called line$().

After that you control what the FORM engine does until you QUIT.

When you quit TBF.EXE it writes the current line$() array to the file called TempForm.TRU. As a result this temporary file now contains your modified program code.

Finally control RETURNS to the editor, which reads the temporary file and assigns its contents to the editor program array. The end result is that your code now appears in the current window of the TB editor.

You can do this process as often as you like and the TB editor will not complain.

As you add more forms to your program it would prudent to SAVE your current file each time. This will not affect the way that TBF.EXE works.

Properties

Now we are going to create another FORM (window) for the same program.

Start FORMS from the TB editor Help for True BASIC menu.

TBF.EXE scans your code to find out how many forms are in your program. It will present you with a list dialog box.

If we select NEW FORM then TBF.EXE will create another form to add to the program. If we select FORM_1 then we can visit the existing form to make changes or additions.

Select NEW FORM then click the OK button.

The screen will show a large gray window with a central menu.

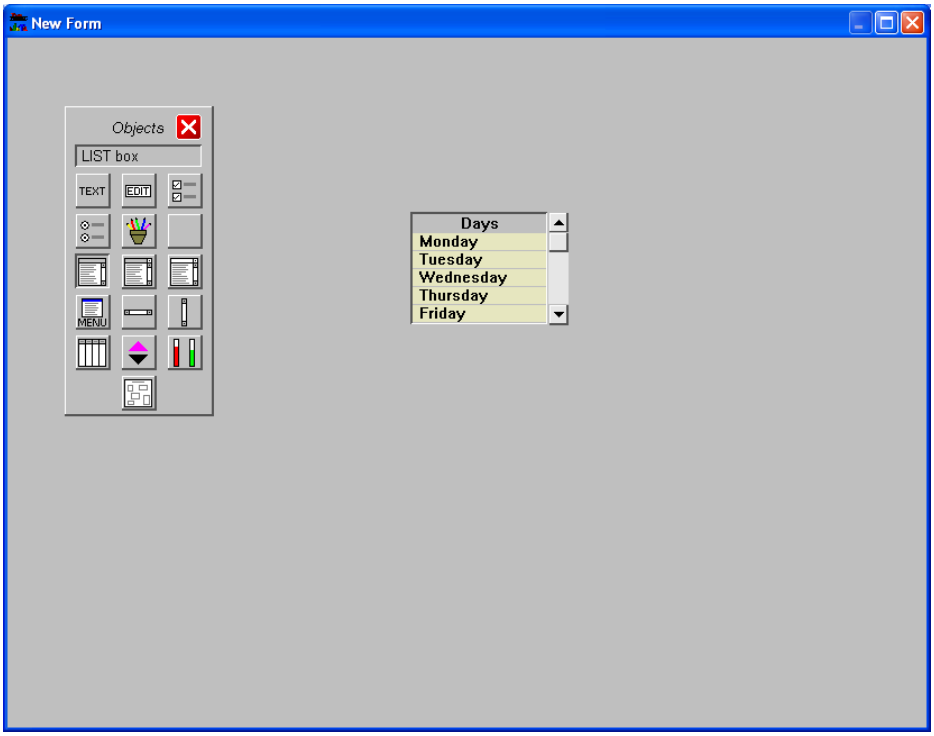
Click the word OBJECTS in the menu to call up the Objects Toolbar.

Now select the third button down on the left. This is the icon for a LIST BOX.

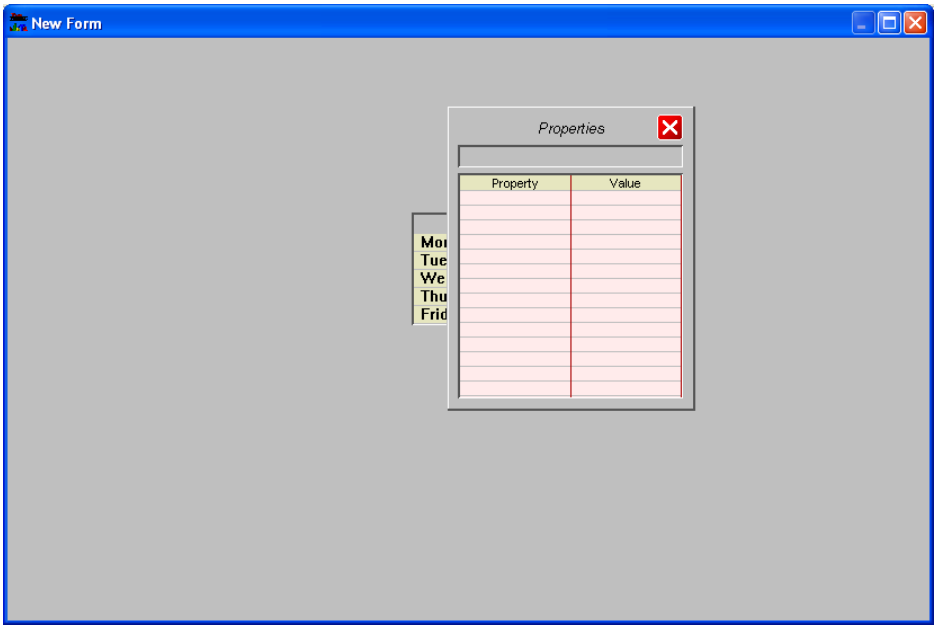
Click somewhere in the middle of the window.

A small list will appear showing the days of the week. There is a built-in default array containing the days of the week. It doesn't look like a normal Windows list because it isn't meant to. It is a demonstration of the color flexibility of the CTX library.

Using the Properties box we are going to customize this list.



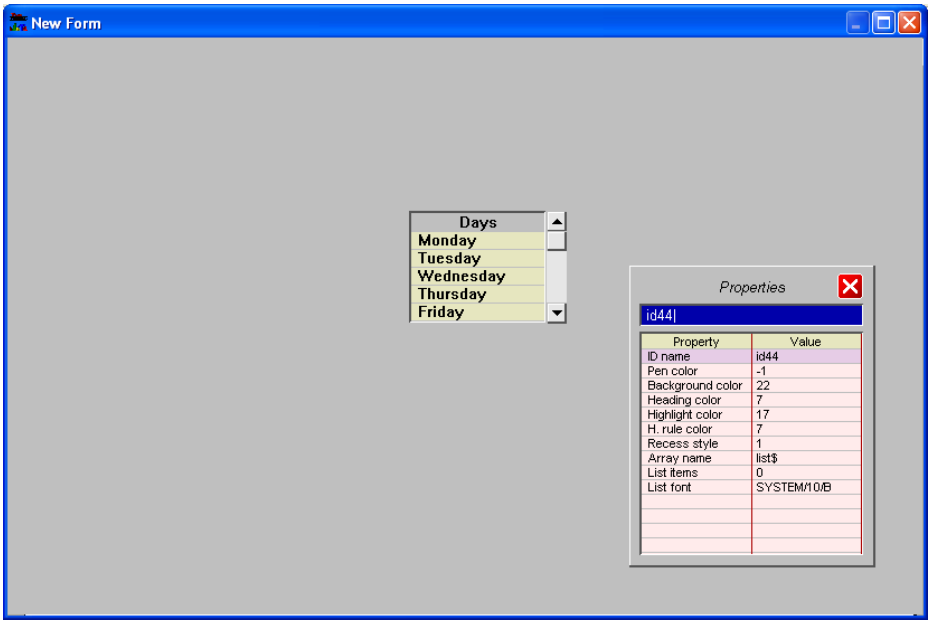
First of all right click the mouse to bring back the menu.
Now select PROPERTIES from the menu.



The empty properties box is probably obscuring your object, so click somewhere near the top left corner of the properties box alongside the heading “Properties”. A black dotted outline will surround the properties box. Click inside and drag the box down to the right bottom corner until the list object can be seen clearly. Both the properties box and the object toolbar can be moved around the screen just by clicking anywhere in the gray margin. Once you have repositioned the box it will re-appear in that position until you move it again.

Now click the list object itself. Notice the dotted outline does not appear.

The properties box will fill up with labels down the left hand side and their current values down the right hand side.



Now click on the first line of the properties list with the label ID name. The current value will be shown in the blue box at the top of the properties box. Normally this will be the letters id followed by a number. This is the weird object identification system described in the introduction. Every object has an ID. You will find it much easier to use descriptive names rather than the default names given by TBF.EXE.

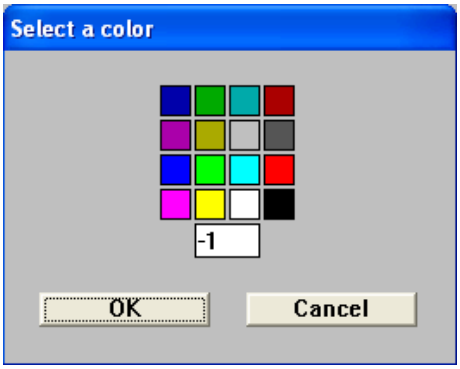
Type the word *list* on the keyboard and it will appear in the box at the top of properties. Now press the RETURN key and the new name will be transferred to the properties below.

You can change as many default values as you like.

Now click on PEN COLOR. In this case (or any label that contains the word COLOR) you will see a COLOR SELECTOR dialog box.

The current pen color is shown in the middle box just above the buttons. The current color of the text is black and the code for black is –1. True BASIC normally uses zero for black but in

Windows zero is reserved for the current window background so –1 is used for black and –2 is used for white.



For the purposes of this exercise we will select bright blue as our pen color.
Click on the blue square (third one down on the left) then click OK.

The text in the list will now be bright blue.

The next four labels also concern color and they work exactly the same way as the example above, so we will skip those and select the label RECESS. This property controls what the object looks like on screen. Windows uses shadows to give the impression of 3D. Our list currently looks like it is a recessed panel sunk into the screen.

The current value is 1 (meaning recessed). Type –1 (meaning proud) at the keyboard then press the RETURN key. The list will now look as if it is standing proud of the screen.

The next property is labelled Array name. The default is list$ but you can use any name you like for the array that contains the items that will appear in the list object.
Click on Array name and type *number$* at the keyboard followed by pressing the RETURN key.
The array name has now changed to number$.

TBF.EXE automatically looks after creating and dimensioning arrays.

The next label is called List Items. This allows us to change the default list i.e. the days of the week to a list of your own making.

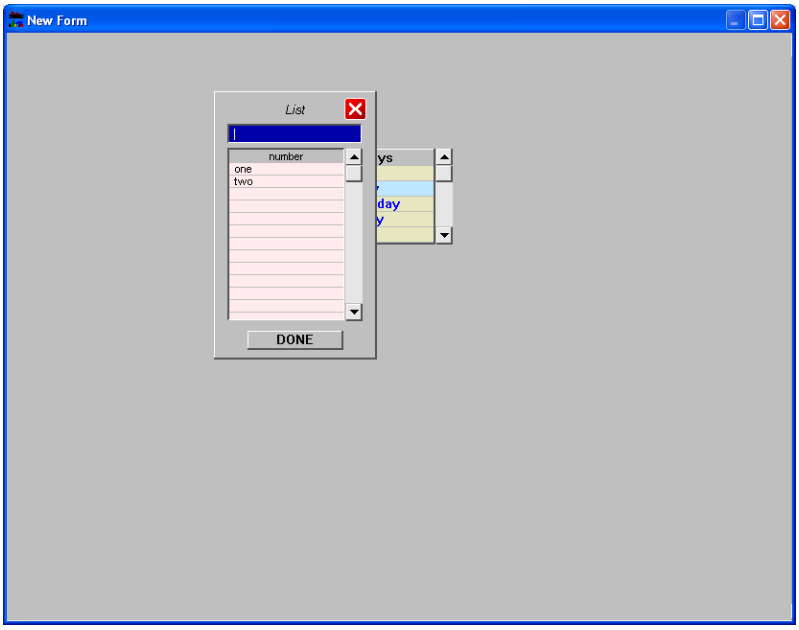
Click on the List items label and a new box will appear with the heading List.
At the keyboard type the word *number* followed by RETURN.

The word number will now appear in the gray heading zone in the list.

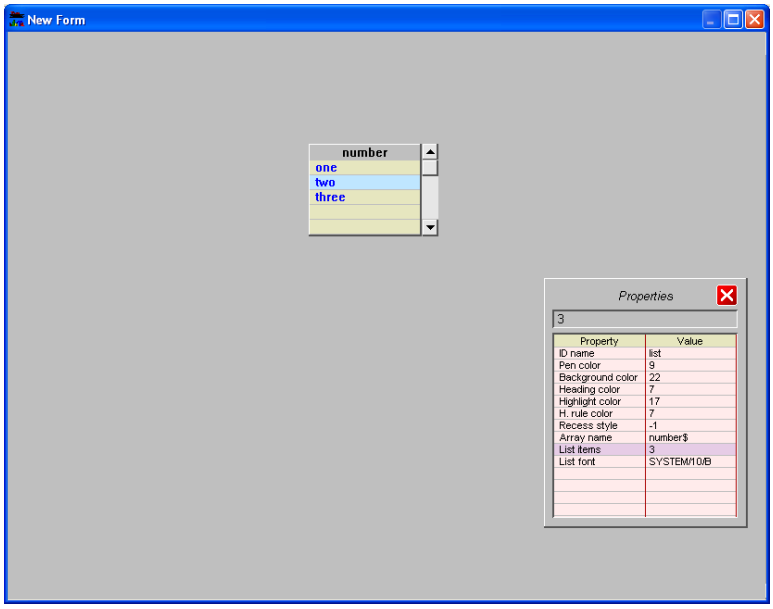
This is how you create headings for all lists including menus,
i.e. the first word you type becomes the heading.

Next type *one* <RETURN> and the word ONE will appear as the first item in the list.
Next type *two* <RETURN> and the word TWO will appear as the next item.
Now type *three* <RETURN> and the word THREE will occupy the next item.

Now click the DONE button. The properties box will re-appear and the list items will show the number 3 because we have entered 3 items.



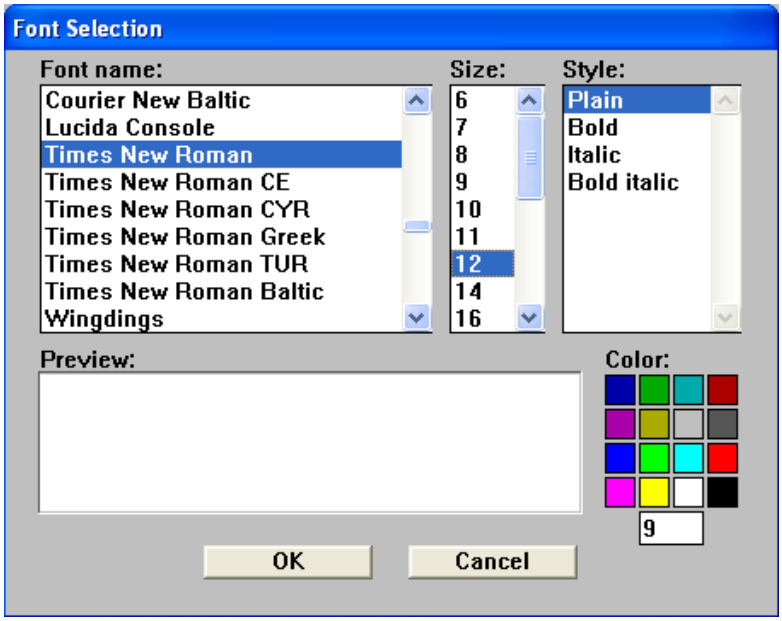
When you click the DONE button it completes the list definition.



Finally we are going to change the font from SYSTEM 10 B to 12 point TIMES.

Click on the label LIST FONT.

A font selector dialog will appear. Any label that contains the word font will produce the font selector.



Click on Times New Roman in the font name list, 12 in the size list and Plain in the style list. You can ignore the color because we have already selected that earlier.

Now click the OK button.

You will see an instant change in the font in the list, reflecting your choice.

This is how you use the properties box.

Each object has a different list of properties including the form itself.

Right click to generate the main menu and to erase the properties box.

Now click WINDOW & MENU.

A new properties box will appear.

Just for demonstration purposes we are going to change the main title in the blue bar at the top of the window.

Click on the label TITLE.

Type the words *Numbers* <RETURN>
You will see that the window title has changed from New Form to Numbers.

Right click to generate the main menu and to erase the properties box.

Now click QUIT and return to the TB editor to see your handiwork in code.

You will see that immediately below the FORM_1 code block there is now a second code block with the title FORM_2

In this second block there will be some code that creates the window and some code that creates the ListBox. You will see that the window itself is created by a call to TrueCTRL and not to CTX, whereas the ListBox is created by a call to CTX. This demonstrates how the two libraries can be used simultaneously. There are technical reasons why windows and their associated menus are created by TrueCTRL, but all other objects are created with calls to CTX. Prior to this code there is an array called number$ and a DATA statement that fills the array with the numerals one to three.

CAUTION: All forms are titled “New Form” when they are first created so you are advised to rename them with a suitable title as soon as they are created using the method above. You will see the importance of this when you next use TBF.EXE.

When TBF.EXE starts up it lists all the current forms in your program and their titles. It can be confusing if more than one form has the same title.

Active

There is an item on the main menu called ACTIVE.

When there is no tick against the word ACTIVE it means the object can be moved and re-sized.

When there is a tick against the word ACTIVE it means the object works just like the real thing; push buttons can be depressed and lists can be selected. In the ACTIVE mode objects cannot be moved or re-sized.

By clicking the word ACTIVE on the menu the tick can be removed or replaced.

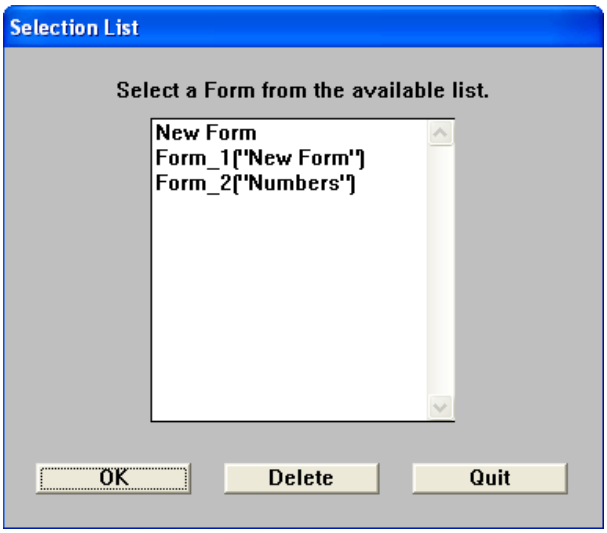
CAUTION: the move and re-size feature with the dotted box only works when the object toolbar is showing. It doesn't work when the properties box is showing.

Menus

The next exercise is to create a menu for a window. Menus appear as a series of headings in the gray band below the blue title bar at the top of a window. When you click on a heading a list of menu items drops down so that you can make your selection. TBF.EXE takes you through the process one heading at a time by treating each heading as a normal list object. This a simpler and more natural way of looking at the task of creating menu structures.

Start FORMS in the normal way.

TBF.EXE scans your code to find out how many forms are in your program. It will present you with a list dialog box.



Now select Form_2("Numbers") from the list of existing forms.
This will allow us to add a menu to the second of the two forms we have designed so far.

The screen will show a large gray window with a small ListBox object in the center.

Use a right click to make the main menu visible.

Select Window & Menu from the menu.

A new properties box will appear.

Before you attempt to build a menu structure you need to have a picture in your mind of how many menu headings you will need and the maximum number of items in the longest list.

Let us assume we will have 3 headings called FILE, WINDOWS and OPTIONS
Let us also assume the longest list will have 4 items. The three lists don't have to have the same number of items; indeed it is very rare for menu lists to be the same length.

Click on the label Menu Headings.
Type the number 3 at the keyboard then press <RETURN>

Now click the next label Max Menu Items.
Type the number 4 at the keyboard then press <RETURN>

A menu box will now appear in the middle of the screen, similar to the list box we used earlier.
The menu box works in exactly the same way.

The title of the menu box will read Menu 1

The first word you type will be the menu heading itself. In this exercise type the word *File* <RETURN>
The word File will now appear in the gray zone at the top of the list.
Now type the following words at the keyboard.
New <RETURN>
Open <RETURN>
Quit <RETURN>

Click the button with the legend NEXT.

The title of the menu box will read Menu 2

As before, the first word you type will be the menu heading itself.
Type the word *Window* <RETURN>
The word Window will now appear in the gray zone at the top of the list.
Now type the following words at the keyboard.

Show Form1 <RETURN>
Show Form2 <RETURN>

Click the button with the legend NEXT.

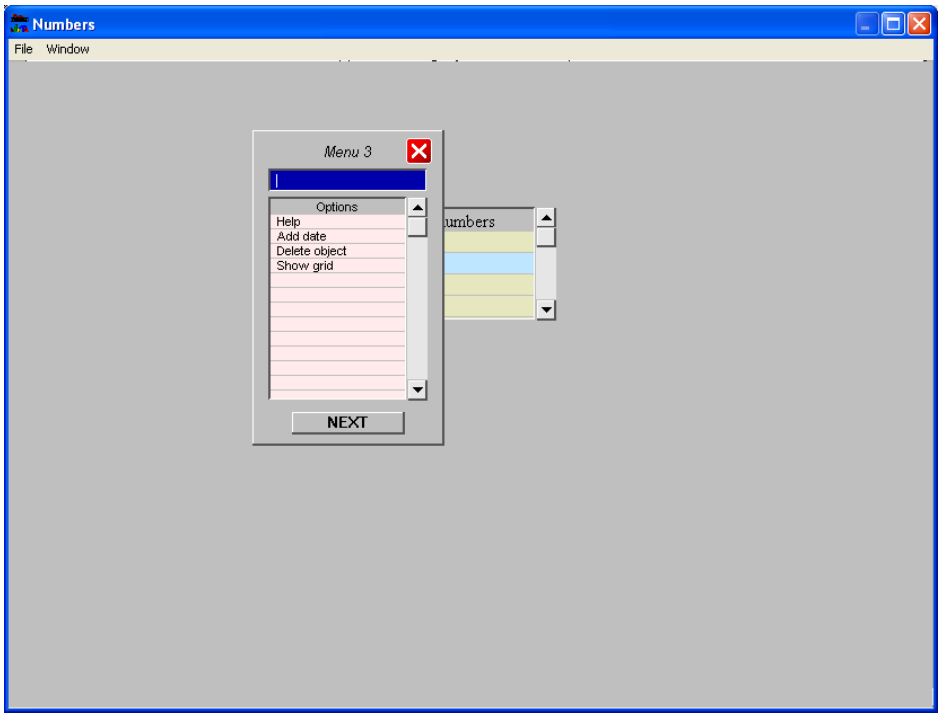
The title of the menu box will read Menu 3.

As before, the first word you type will be the menu heading itself.
Type the word *Options* <RETURN>
The word Window will now appear in the gray zone at the top of the list.
Now type the following words at the keyboard.
Help <RETURN>
Add date <RETURN>
Delete object <RETURN>
Show grid <RETURN>

Click the button with the legend NEXT.

The menu box will now disappear and the properties box will return.

You will notice that your form now has a gray band at the top with 3 menu headings.
Click on these headings in turn to see the drop down list of menu items.



Right click to generate the main menu and to erase the properties box.

Now click QUIT and return to the TB editor to see your program code.

CAUTION: If you accidentally click the NEXT button too soon, or you make some other foolish mistake, just click the CLOSE button (white cross on a red background) and go back to the properties box. Now start over and do the whole exercise again. This is the penalty for making errors.

A final word about menus. The menu structures we have worked with so far are simple menus, i.e. each has a heading followed by a list of items. For most practical purposes this is all you need. However, TrueCTRL allows you to construct hierarchical menus, i.e. each item in the menu list can have a child menu attached to it. Likewise the child menu items can each have further child menus attached to them. TBF.EXE does not generate hierarchical menus.

If you look at your program code in the TB editor you will see that there is a whole bunch of sub-routines waiting for you to add custom code to deal with menu selections. The names of these sub-routines have been modified slightly to remove spaces in menu item names. For example, one of the menu items is Show Form1. This has been modified with an underscore in the sub-routine name:
SUB action_menu_Show_Form1

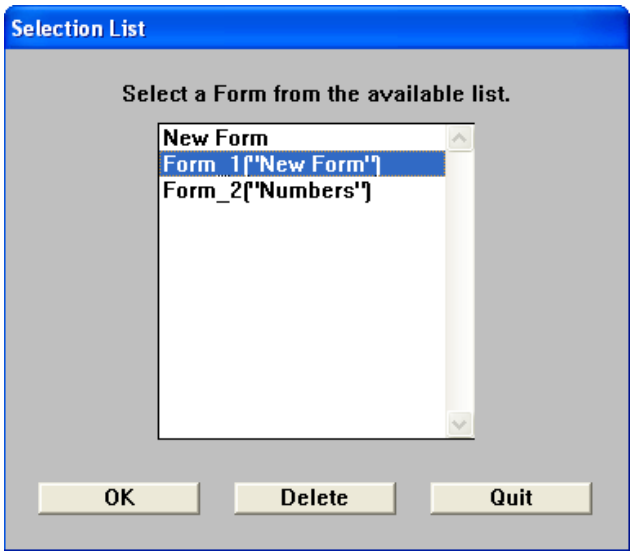
If the space had been left in then True BASIC would signal an error if you tried to run the program.

Right Click Menus

Creating a right click menu object is very similar to creating a normal window menu except that there is only one heading so the task is much simpler.

Start FORMS in the normal way.

TBF.EXE will present you with a list dialog box.



Now select Form_1(“New Form”) from the list of existing forms.
This will allow us to add a menu to the first of the two forms we have designed so far.

The screen will show a large gray window with three objects.

Use a right click to make the main menu visible.

Select Objects from the menu.

A right click menu is not a window feature like a normal menu. It is an object and can be seen on the toolbar on the left, fourth row down.

Select the right click menu icon.

The properties box will open immediately.

The earlier advice concerning menus still holds true. Before you attempt to build a menu you need to have a picture in your mind of how many menu items the right click menu is going to have. You also need to think of a suitable heading even though this is never displayed. It is just an internal way of recognizing the menu.

Let us assume we will have 4 items called UP, DOWN, LEFT and RIGHT. For convenience we will use the word DIRECTION as the heading.

Click on the label Menu Items. (Ignore the current value shown).

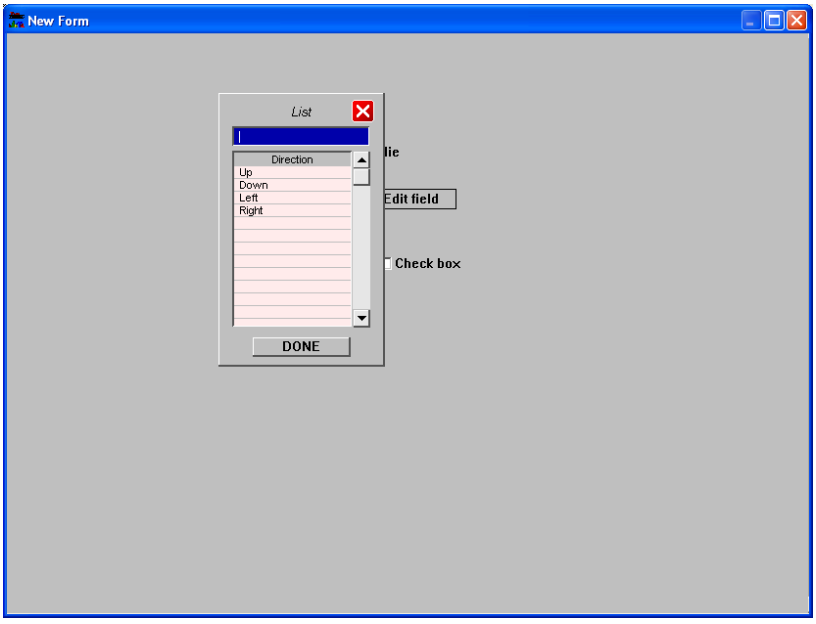
A list box will immediately appear in the middle of the screen, similar to the list box we used earlier.

The title of the list box will read List

The first word you type will be the menu heading itself. In this exercise type the word *Direction* <RETURN>

The word Direction will now appear in the gray zone at the top of the list.
Now type the following words at the keyboard.
Up <RETURN>
Down <RETURN>
Left <RETURN>
Right<RETURN>

Click the button with the legend DONE.



As soon as you press DONE the program will count up the number of menu items and will show this in the properties box.

This is how you create right click menus.

If you want to see the right menu in action then you are out of luck. TBF.EXE already has a right click menu so making a right click will show the usual main menu. A window (form) can only have ONE right click menu. When you think about this it makes sense. If you had more than one right menu, how would the computer know which one you wanted.

To see the right click menu in action you must first quit TBF.EXE. This is done by making a right click to produce the main menu. Now select QUIT.

Instantly you will return to the True BASIC editor where you can see that your program contains the necessary code to produce the right click menu. You can also see a list of sub-routines where you must add some custom code to deal with any right menu selection.

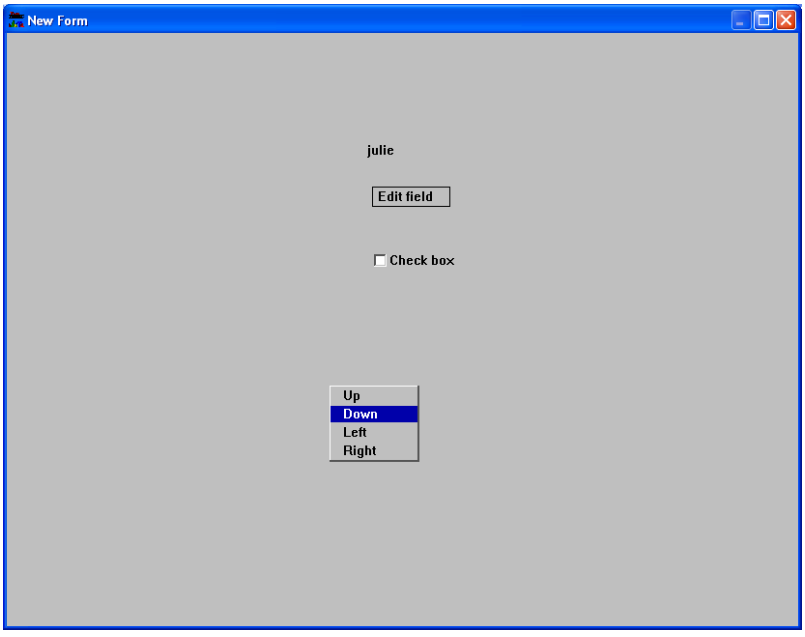
Notice that in your program, the block of code associated with Form_1 is now below the code for Form_2. The rule is that whatever form you worked on last appears at the bottom of the Form blocks.

Select RUN from the TB editor RUN menu.

You program will show the original three objects.

Now make a right click anywhere in the window and the menu you just created will appear. Check that the highlight works by running your mouse over the menu.

Press the ESC key to quit and return to the editor.



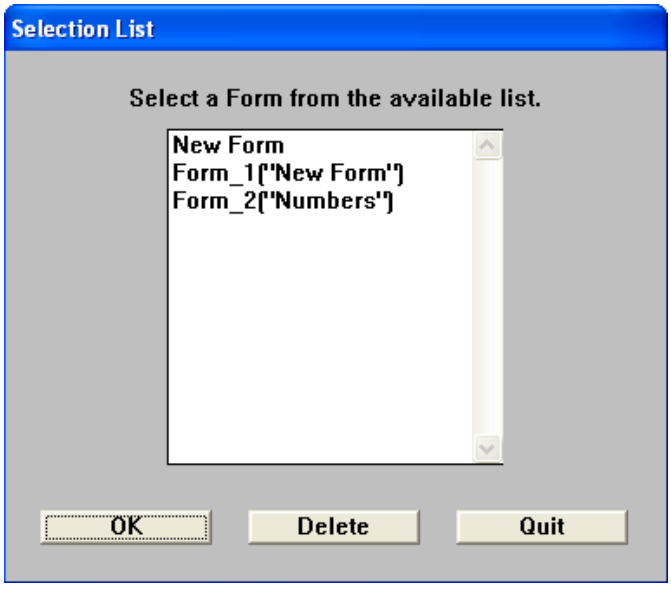
TBF.EXE does not generate hierarchical menus, however the CTX library module does allow these complex structures – see the RmenuDemo.TRU program for a demonstration of how this is done with right click menus.

Group Box

The TrueCTRL groupbox is merely decoration. It performs no useful function other than to surround other objects with a line to give them the appearance of a group. On the other hand the CTX groupbox has some very useful and unique features. The purpose of the next exercise is to demonstrate those unique features. Basically a CTX groupbox makes a real group out of all the objects contained within it. When the groupbox is shown, all the objects inside are shown as well. When you hide a groupbox, all the objects inside are hidden. When you move a groupbox using the dotted rubber box system, then all the objects inside move with it. Even though the objects act as a group, they can also act individually. For example, you can hide an individual object without affecting any of the others. You can also move individual objects around inside the groupbox without affecting other objects. You can also re-size objects in a groupbox. In the TBF.EXE program the object toolbar, the properties box and the list/menu box are all examples of a groupbox.

The rule for using the groupbox is that you must create the individual objects first, and then you create the groupbox to physically surround them. Any object that exists within the boundaries of the groupbox will be included in the groupbox.

Start FORMS in the normal way.



Now select Form_2(“Numbers”) from the list of existing forms.
This will allow us to add a menu to the second of the two forms we have designed so far.

The screen will show a large gray window with a small ListBox object in the center.

Use a right click to make the main menu visible. Select Objects from the menu.

First of all let us create a few more objects to put inside the groupbox.

Select the first icon (Static Text) and click just above the numbers list in the center of the window. The default image is a CLOSE button.

Next select the image icon (middle icon second row) and click alongside the static text object you just created. Move the image until it is aligned with the right hand edge of the list.

Now click the text object to give it a dotted outline and resize it so that it extends from the left hand edge of the numbers list up to the edge of the image object.

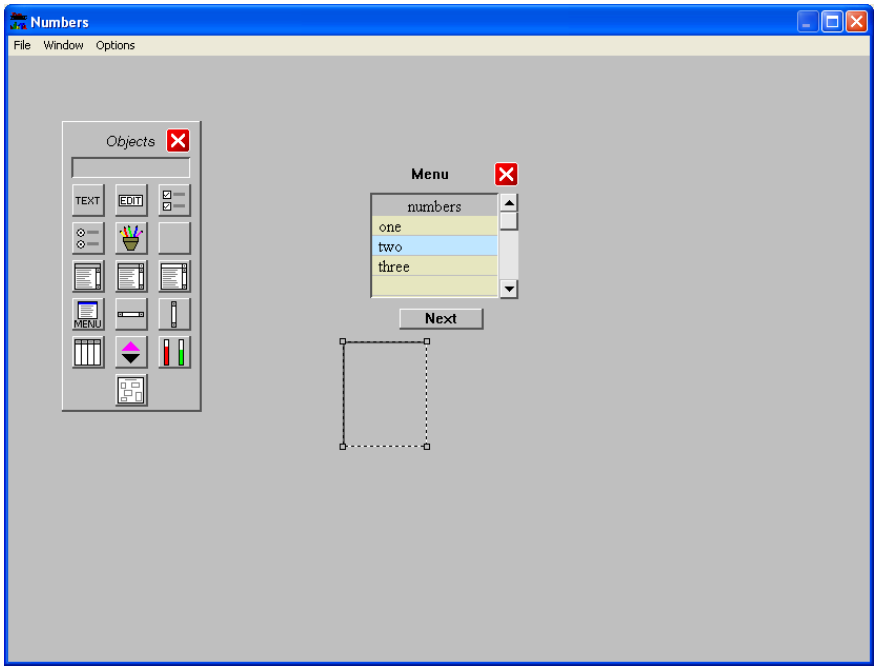
Next select the push button icon (second row right hand) and click below the numbers list. Move and resize the push button so that it fits between the left and right edges of the bottom of the numbers list.

Now let us customize these objects. Right click for the main menu and select Properties.

Click on the static text object.
In the properties box click on TEXT and type Menu at the keyboard.
In the properties box click on JUSTIFY and type 1 (centralize) at the keyboard.

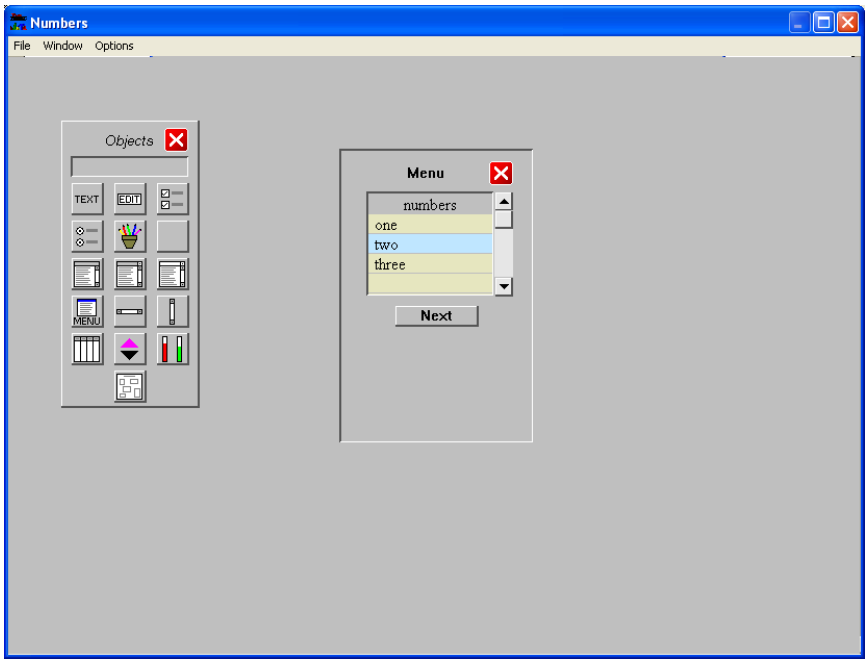
Click on the push button.
In the properties box click on LEGEND and type NEXT at the keyboard.

Right click for the main menu and select the groupbox icon (bottom). Click about one inch below the push button and slightly to the left. A blank plate will appear.



Click the groupbox to make a dotted outline then drag the top right corner to enclose all the objects on the screen. Repeat this process to re-size and move the groupbox until the objects fit nicely inside.

This is how you make a group box.

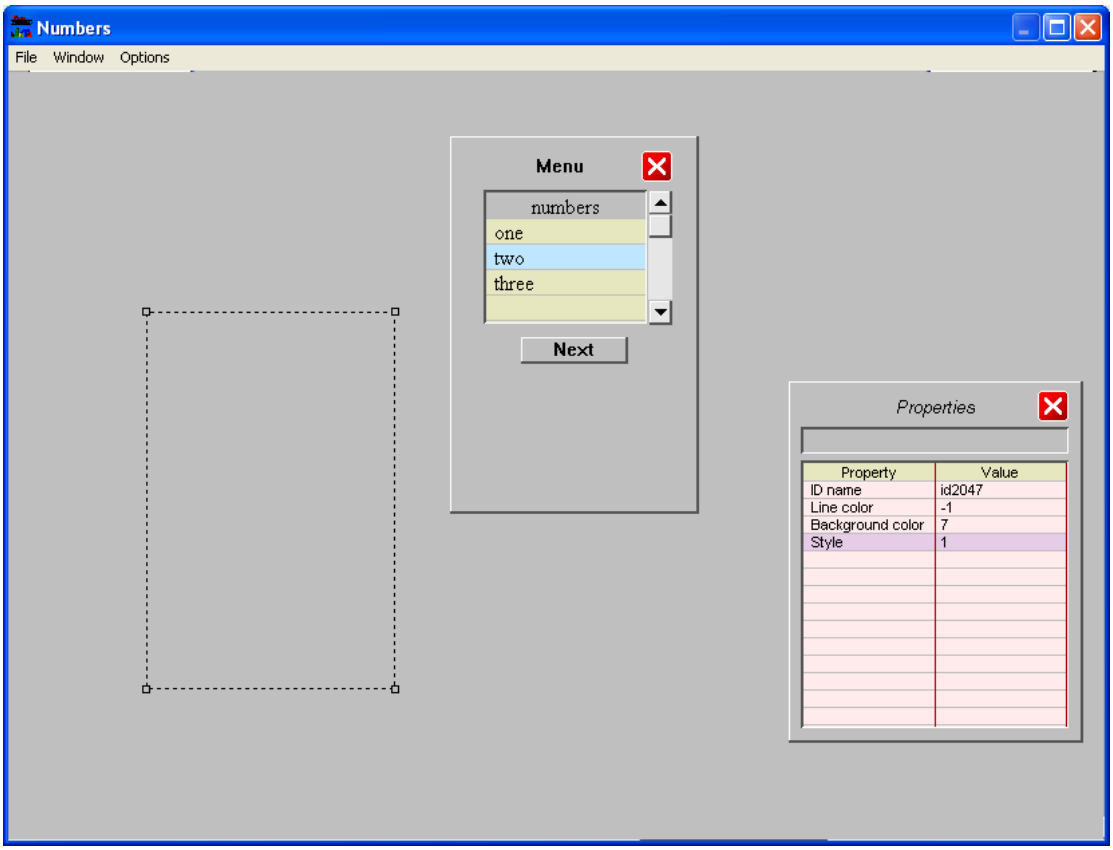


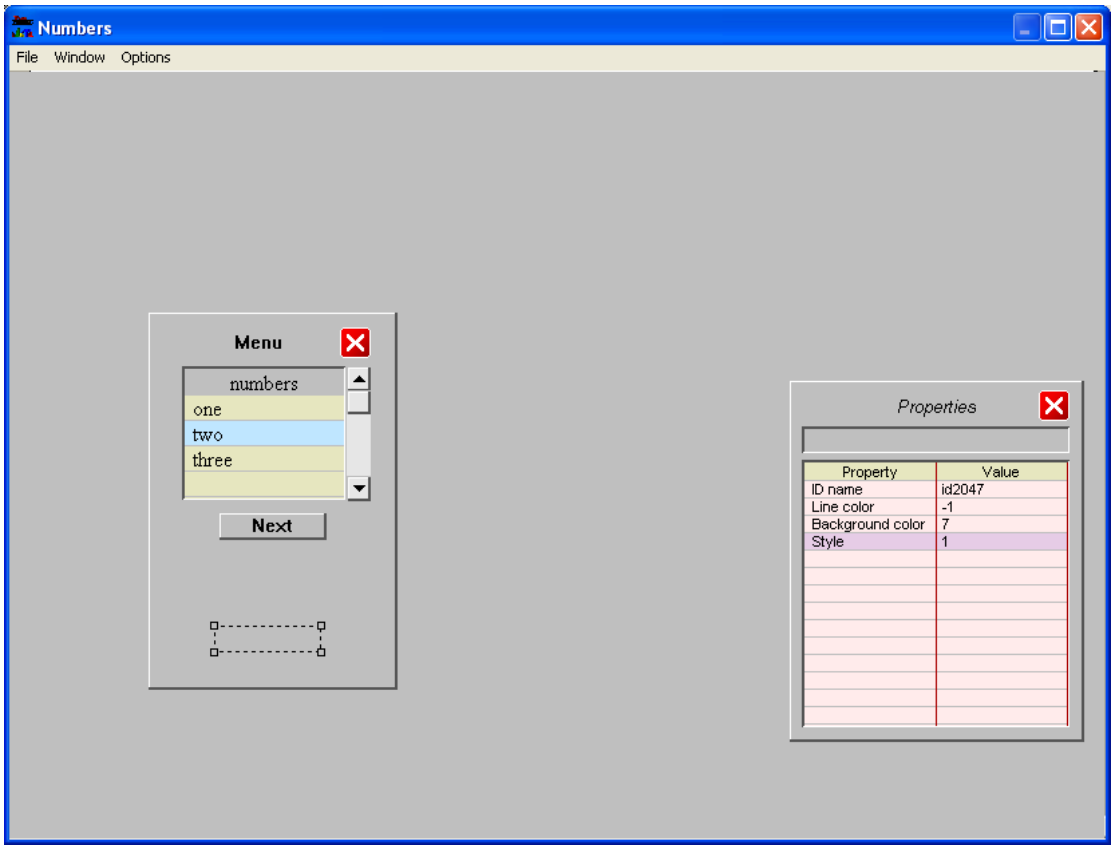
You can change the appearance of the groupbox by selecting Properties from the right click menu, and then click anywhere other than an object in the gray area of the box.

Now select STYLE from the properties list and type –1 at the keyboard. This will make the groupbox stand proud of the screen by re-arranging the shadows.

Now try moving the groupbox to a new location. First do a right click then untick the ACTIVE option. Click anywhere inside the groupbox to produce the dotted rubber box. Now you can move the groupbox around the screen and it will take all the objects with it.

You can also move the objects inside the groupbox. Click on the NEXT button and move it up and down but keep it in the groupbox.





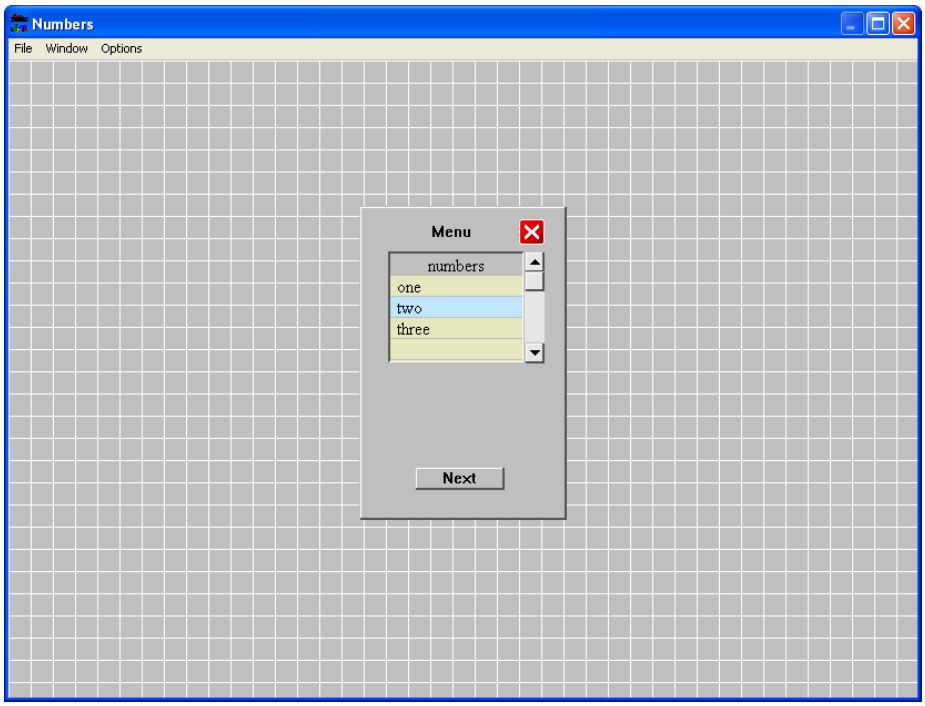
Right click and select QUIT from the menu to save the new objects you have created.

Grid

One of the problems when you are working with multiple objects is aligning edges to give your form the appearance of neatness and symmetry. To assist you in this task, the main menu has an item called GRID.

If you click GRID then a tick will appear alongside the word GRID. The screen will show a grid pattern of white lines on a gray background like a sheet of graph paper.

Click again on the word GRID and the grid lines will disappear. The tick mark will also be erased.



Grid Pitch

The next item on the main menu is GRID PITCH. This option allows you to change the pitch of the grid. The units are pixels. A simple dialog requests you to enter the grid pitch. The default is 20 pixels. As an exercise enter 40 and view the results.

Snap

The main right click menu has an item called SNAP. Snap is used in conjunction with GRID. The purpose of snap is to align your current object with the nearest grid intersection. In other words, when you create a new object one of the corners will coincide exactly with a vertical and horizontal grid line. Snap ONLY works when you create an object. It does NOT work if you move or re-size an object with the dotted rubber box. It is a convenient and easy way to align objects.

If you click SNAP then a tick will appear alongside the word SNAP. This means the snap feature is switched on.

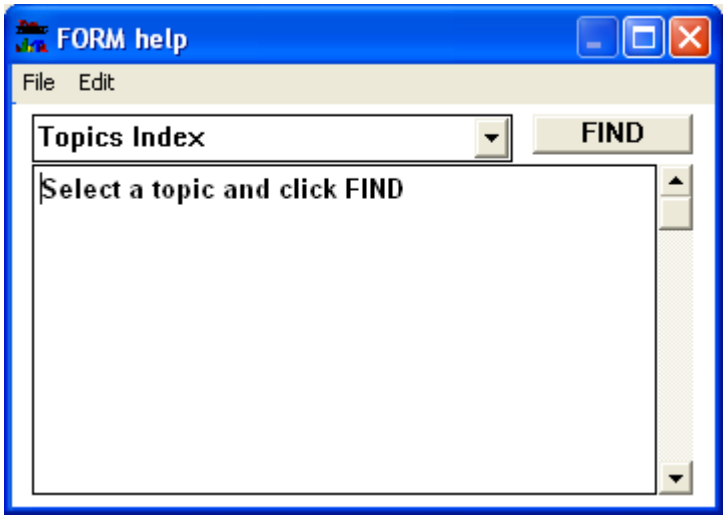
Click again on the word SNAP and the tick mark will be erased. This means the snap feature is switched off.

Help

You cannot be expected to remember all the detail in these instructions. There will be many times when you will want to refer to the text for further guidance. Even very experienced True BASIC users often refer to the Manual to make sure they use the right syntax and the right list of parameters.

TBF.EXE has a built-in HELP window that can be displayed by selecting HELP from the right click menu.

Normally, if you select help from main menu, then the help window shows an alphabetical index that can be seen by clicking the small arrow button alongside the Topics Index.



Click on the topic you want, and then click the FIND button.

The HELP index also contains guidance on how to create your own notes and tips and how to include them in the help file.

If you have the properties box open and you have selected an object then if you select HELP from the menu the topics index will show you the appropriate section related to the object you are working with.

Index of Properties

Static text

Title	Default	Comments
ID name	idN	N is a number given by the program
Text	Static tex	
Pen color	-1	Black
Background color	7	gray
Recess style	-99999	1=recessed:-1=proud:0=outlined
Dotted outline	0	0=off: 1=on
Justify	0	0=left: 1=center: 2=right
Text font	System/10/B	

Check Box

Title	Default	Comments
ID name	idN	N is a number given by the program
Text	Check box	
Pen color	-1	Black
Background color	7	gray
Dotted outline	0	0=off: 1=on
Justify	0	0=left: 1=center: 2=right
Text font	System/10/B	

Radio button

Title	Default	Comments
ID name	idN	N is a number given by the program
Button Text	Radio button	
Group ID name	group	
Group array name	Radio$	
Pen color	-1	Black
Background color	7	gray
Dotted outline	0	0=off: 1=on
Justify	0	0=left: 1=center: 2=right
Text font	System/10/B	

Group box

Title	Default	Comments
ID name	idN	N is a number given by the program
Pen color	-1	Black
Background color	7	gray
Recess style	1	1=recessed: 0=outline: -1=proud

Edit field

Title	Default	Comments
ID name	idN	N is a number given by the program
Initial Text	Edit field	
Pen color	-1	Black
Background color	7	gray
Recess style	0	0=outlined: 1=recessed: -1=proud
Max characters	0	Only use for auto advance
Check contents	0	0=off: 1=on
Text font	System/10/B	

Push button

Title	Default	Comments
ID name	idN	N is a number given by the program
Button Text	Button	Auto centralized
Pen color	-1	Black
Background color	7	gray
Toggle action	0	0=off: 1=on
Justify	0	0=left: 1=center: 2=right
Text font	System/10/B	

DATA table

Title	Default	Comments
ID name	idN	N is a number given by the program
Pen color	-1	Black
Background color	22	Sand
Heading color	7	gray
Highlight color	17	Light green

V. rule color	6	ochre
H. rule color	7	gray
Recess style	1	1=recessed: 0=outline: -1=proud
V. scroll	1	1=show: 2=hidden
H. scroll	0	1=show: 0=hidden
Array or filename	Form_table.csv	Must be CSV structure
Table font	System/10/B	Recommended Arial/8/P

List Box

Title	Default	Comments
ID name	idN	N is a number given by the program
Pen color	-1	Black
Background color	22	Sand
Heading color	7	gray
Highlight color	17	Light green
H. rule color	7	gray
Recess style	1	1=recessed: 0=outline: -1=proud
Array name	List$	Must be dimensioned [0:n]
List items	0	Number inserted automatically
List font	System/10/B	

List Button

Title	Default	Comments
ID name	idN	N is a number given by the program
Pen color	-1	Black
Background color	22	Sand
Heading color	7	gray
Highlight color	17	Light green
H. rule color	7	gray
Recess style	1	1=recessed: 0=outline: -1=proud
Array name	List$	Must be dimensioned [0:n]
List items	0	Number inserted automatically
List font	System/10/B	

List Edit

Title	Default	Comments
ID name	idN	N is a number given by the program
Pen color	-1	Black
Background color	22	Sand
Heading color	7	gray
Highlight color	17	Light green
H. rule color	7	gray
Recess style	1	1=recessed: 0=outline: -1=proud
Array name	List$	Must be dimensioned [0:n]
List items	0	Number inserted automatically
List font	System/10/B	

Right Menu

Title	Default	Comments
ID name	idN	N is a number given by the program
Pen color	-1	Black
Background color	-2	White
Recess style	1	1=recessed: 0=outline: -1=proud
Highlight color	1	Dark blue
Menu list items	0	Number inserted automatically
Menu font	System/10/B	

Elevator buttons

Title	Default	Comments
ID name	idN	N is a number given by the program
Bezel color	-1	Black
Bezel face color	7	gray
Style	0	0=roundrect: -1=square
Arrow color	13	Magenta

Bar Scale

Title	Default	Comments
ID name	idN	N is a number given by the program
Outline color	-1	Black
Background color	7	gray
Positive bar color	9	Blue
Negative bar color	9	Blue
Style	1	1=recessed: 0=outline: -1=proud
Low scale range	1	
High scale range	100	
Mouse increment	1	

Horizontal scroll

Title	Default	Comments
ID name	idN	N is a number given by the program
Body color	17	Silver gray
Start range	1	
End range	100	
Increment	1	Increment=arrow buttons
Page	10	Page=thumb to arrow area

Vertical scroll

Title	Default	Comments
ID name	idN	N is a number given by the program
Body color	17	Silver gray
Start range	1	
End range	100	
Increment	1	Increment=arrow buttons
Page	10	Page=thumb to arrow area

Images

Title	Default	Comments
ID name	idN	N is a number given by the program
Image file	Image.bmp	Must be BMP or JPG
Location	LB	LB=left bottom: RT=top right

Window

Title	Default	Comments
ID name	idN	N is a number given by the program
Iconize	0	0=No: 1=Yes
Title	New Form	
Menu headings	0	
Max items	0	

For additional information about TBF.EXE objects, refer to the CTX Manual. This manual contains extensive details of all objects and the sub-routines related to them.

You will see that your program automatically calls the TDX library module. For further details of how to make calls to this library refer to The TDX Manual where you will find full details of the extended range of dialog boxes.

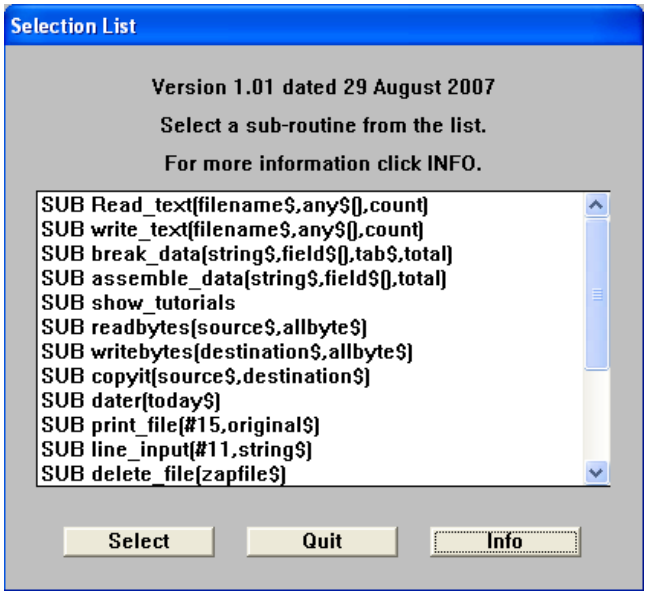
Utilities

You will probably find that in many of your programs you will want to perform the same sort of tasks over and over again. For example, you will almost certainly want to read a text file or a data file to bring data into your program, and it is equally likely that you will want to write text or data to a file to save that data for future use. All programs generated by TBF.EXE already contain two such utility routines:
SUB Append_file
DEF Valstr

The UTILITIES option on the main right click menu allows you to add a number of useful routines to your program.

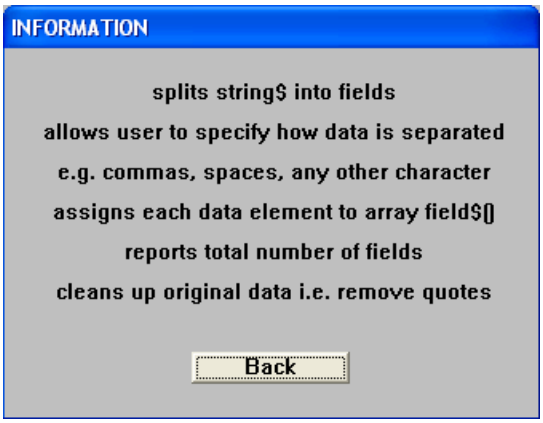
Right click to show the main menu then select UTILITIES.

A list dialog box will show you a list of all the sub-routines that can be added automatically to your program.



There are three buttons below the list; SELECT, INFO and QUIT
Click on any sub-routine in the list then:
If you click the SELECT button the routine will be added to the end of your program.
If you click the QUIT button then no routines will be added to your program.
If you click the INFO button then another dialog box will appear that gives you some brief information about the routine and what it does. Click BACK to return to the selection list.

Note the current version number of the TBF.EXE program is shown at the top.



Any sub-routine you select will be added to the end of your program below the section marked UTILITIES. You will be responsible for writing the CALL to this routine and for supplying the appropriate parameters. The routine itself contains notes about the parameters that are required. Ignore them at your peril.

Once a routine has been added to the Utilities section it cannot be removed automatically. However, you can delete unwanted routines manually or you can just leave them there because they will not affect the way your program works.

It is very likely that over time you will acquire a number of useful sub-routines that you use over and over in a variety of programs. These personal sub-routines can be added to those supplied with TBF.EXE.

The file containing the sub-routines is called FORM_UTILITIES.TXT
This file can be accessed with the TB editor or with NOTEPAD.
Just add your own routines to the end of the present list and TBF.EXE will look after everything else.

Make sure you add brief notes preceded by a REM: statement in a similar way to the existing routines. These notes will appear in the INFO box. Don’t assume these notes are a waste of time. In time you will almost certainly forget why you wrote the routine, what it does and how it works. The INFO dialog box has a limit of 10 display lines so keep your description brief.

Make sure you explain what the parameters are, and precede these notes with the comment character (!) in the same way as the existing sub-routines.

It is good practice to document your routines in this way. You will eventually be grateful for this advice. At some time in the future you will look back on some long forgotten or half remembered program and you won’t have a clue how it works unless you have these little notes to help you.