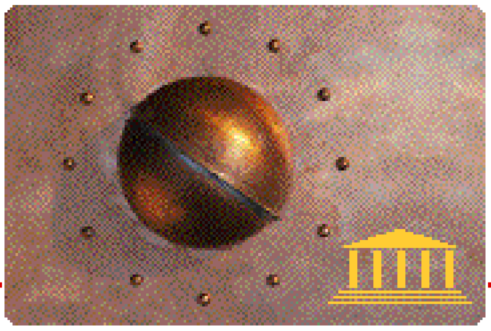

Secret Internals

by Christopher L. Sweeney

Chief Developer, True BASIC Inc.

Handy Undocumented True BASIC Features You Can Use

Introduction

While the True BASIC Language System is not rife with undocumented features, there are a few items here or there that we have written in to provide a bit more speed or convenience for some specific purpose. In this first in a series of occasional articles, I will begin to share some of those routines which might be generally useful to True BASIC users. The following routines are available in the True BASIC Bronze, Silver, and Gold Editions for Macintosh, Windows, and OS/2, unless otherwise noted.

Quick and Dirty Dialog

We found it useful to be able to display a dialog box with a simple line of text and an “Ok” button quickly and easily without going through the TrueDials library or the TBD routine directly, for debugging purposes. This routine has much less overhead than the others and can be especially useful in low-memory situations.

We have a general-purpose internal routine with many options in True BASIC that we use for various debugging and other low-overhead purposes. It is defined as:

```
SUB WInfo(msg$,value$)
```

You can raise the quick-and-dirty dialog by setting `msg$` to “dialog” and `value$` to the text you wish to display in the dialog and calling WInfo.

```
LET msg$ = "dialog"
LET value$ = "The quick brown fox jumped over the lazy
dog."
CALL WInfo(msg$,value$)
```

The dialog box is modal, so WInfo won’t return until the dialog box is put away by clicking “Ok”.

A Modern Window #0

In older versions of True BASIC (2.x through 4.x) if you needed to switch to a logical window you were sure to have, you could switch to **window #0**.

You can still do this in modern versions of True BASIC, but there is one problem: logical window #0 is in the default physical output window, so using a

`WINDOW #0` statement will show that window. If you are working with a different window of your own creation, this may not be handy or elegant.

In True BASIC, we ran into this problem when closing a logical window in a physical window. We needed to select another logical window in the same physical window to be active, but window #0 was not usually a good choice. So we wrote a routine to find the first logical window opened and still open in a given physical window and use that. It is the equivalent of a `WINDOW #x` command, where `x` is the file channel of the logical window chosen.

You can access this feature by using the WInfo routine described in the previous section. Set `msg$` to "set first logical window" and `value$` to the number of the physical window in question, converted to a string, of course.

```
LET msg$ = "set first logical window"
LET value$ = str$(pwid)      ! pwid from TC_Win_Create
                             ! or Object routine.
CALL WInfo(msg$,value$)
```

You must be careful about one thing: the routine assumes that there is, in fact, a logical window open in the physical window. If it finds none, it returns without taking any action. If you have used True Controls exclusively when opening and managing your windows, there will always be at least one logical window open in every physical window.

How Deep are your Colors?

Sometimes when writing a True BASIC program, it is handy to know what the color depth of your graphics card is, which is to say, how many bits are used to generate the color for each pixel of your display. For example, you may want to load a different bitmap or mix colors with `SET COLOR MIX` differently based on the color depth.

There are several undocumented options to the `OBJM_SYSINFO` method and `TC_GetSysInfo` routine, one of which will help in this instance. Use the attribute "BITS PER PIXEL". The number of bits per pixel will be returned in `values(1)`:

```
LIBRARY "TrueCtrl.trc"
DIM values(0)
LET attr$ = "BITS PER PIXEL"
CALL TC_GetSysInfo(attr$,"",values)
LET bits_per_pixel = values(1)
```

Common values will be 8 for 256 colors, 15 or 16 for 32K or 64K colors ("Thousands" on the MacOS), and 24 for 16M colors ("Millions" on the MacOS).

Macintosh Architecture

Another bit of information it is sometimes handy to know when writing True BASIC programs for the Macintosh is whether you are running on a 68000-based machine or a PPC-based machine. (This is particularly helpful when calling C routines using the Gold Edition, so the technique described below is in fact documented in the Gold Edition documentation.)

You can determine this by using the `OBJM_SYSINFO` method or `TC_GetSysInfo` routine with the “ENV” attribute and asking for the value of “ISA” (which stands for “Instruction Set Architecture”):

```
LIBRARY "TrueCtrl.trc"
DIM values(0)
LET attr$ = "ENV"
LET envvar$ = "ISA"
CALL TC_GetSysInfo(attr$,envvar$,values)
IF pos(envvar$,"PPC") > 0 THEN
      ! Machine is PPC-based...
ELSE
      ! Machine is 68000-based...
END IF
```

If the final value of `envvar$` contains the string “PPC”, then the machine is PPC-based. If it contains instead the string “68K”, then it is 68000-based. (The final value of `envvar$` will be either “ISA|PPC” or “ISA|68K”

Note that this is similar to finding the pathname of the Preferences folder or Temporary folder on the Macintosh, which uses the same method, subsituting “PREFFOLDER” or “TMP” for “ISA” and returning the appropriate pathname after the environment variable name and delimiter in `envvar$`.

Finding the Windows System Directories

When writing True BASIC programs for the Windows family of operating systems, it may occasionally be useful to know the pathnames of the WINDOWS and SYSTEM directories. While these are often C:\WINDOWS\SYSTEM for Windows 3.1 through Windows 98 and C:\WINNT35\SYSTEM32 or C:\WINNT\SYSTEM32 for Windows NT 3.5 and 4.0, respectively, they may not be, so it is best not to count on the common possibilities for these pathnames.

You can determine the actual WINDOWS and SYSTEM directory pathnames by using the OBJM_SYSINFO method or TC_GetSysInfo routine with the “SYSTEM DIRECTORIES” attribute. The result will be the SYSTEM directory, then the WINDOWS directory, separated by the current delimiter (“|” by default):

```
LIBRARY "TrueCtrl.trc"
DIM values(0)
LET attr$ = "SYSTEM DIRECTORIES"
LET sysdirs$ = "" ! result will be placed in this
variable
CALL TC_GetSysInfo(attr$,sysdirs$,values)
```

For example, if I run this on my development machine, `sysdirs$` will be “D:\WINNT\SYSTEM32|D:\WINNT” when the call returns.

June 2, 1999