Apple II Technical Notes

Developer Technical Support

Apple IIGS
#73: Using User Tool Sets

Revised by: Dave “flag bits” Lyons July 1991
Written by: Dave Lyons November 1989

This Technical Note explains how to write a user tool set and why writing a user tool set is better than stealing a system tool set number.

Changes since January 1991: Expanded recommendation on where to keep user tool set files on disk and clarified `SetTSPtr` information.

The *Apple IIGS Toolbox Reference* describes system tool sets, which are usually called through the system tool dispatcher vectors 1 ($E10000) and 2 ($E10004).

There are 255 possible system tool set numbers (1 through 255). All of these are reserved for definition by the system. If your program is “borrowing” a system tool set number, please feel guilty and switch over to the user tool set numbers. There are 255 of them too, and they’re called through user tool dispatcher vectors 1 ($E10008) and 2 ($E1000C). All 255 user tool set numbers are available for the current application to use as it chooses. (Desk accessories are forbidden to use user tool sets.)

Of the four tool dispatcher vectors, only the first one ($E10000) has received a lot of publicity. $E10008 works just like $E10000, except that it passes control to a user tool set instead of a system tool set.

The second vector of each pair ($E10004 and $E1000C) works just like the first, except that one extra RTL address must be pushed onto the stack after any parameters are pushed. This way you can have a subroutine to do some or all of your toolbox dispatching, and that subroutine can do extra processing before or after the tool call, or both.

How Can I Write a User Tool Set?

Appendix A of *Toolbox Reference*, Volume 2, shows how to write a user tool set. Your tool set's Work Area Pointer is a four-byte value you can set with SetWAP and get with GetWAP. The WAP value is already loaded into the Y and A registers every time one of your tool set's functions gets control. The traditional use for the WAP is to keep track of an area of memory owned by your tool set.

If you do use the WAP in a conventional way, your xxxStatus function should return TRUE if the WAP is nonzero; your xxxStartUp function should set the WAP to a non-zero value pointing to some memory space you own (provided by the caller, or allocated with NewHandle using a memory ID provided by the caller); and your xxxShutDown function should set the WAP back to zero.

Since the X register contains the tool set and function number when one of your functions gets control, it is not necessary for a tool set to be written to be used as a predetermined user tool set number. At execution time, your tool set can compute the proper error codes and values to send to GetWAP and SetWAP.

Note: At the bottom of page A-8 of the *Apple IIGS Toolbox Reference*, Volume 2, "lda #$90" should read "lda #$8100" for version 1.0 prototype. On page A-10, the figure should show **two** RTL addresses (6 bytes) on the stack.

ToStrip and ToBusyStrip Vectors

These two vectors are for tool sets to jump to when a function exits.

```
ToBusyStrip   $E10180
ToStrip       $E10184
```

Inputs: X = error code (0 if no error)

Y = number of bytes of input parameters to strip

When your function is ready to exit, set up the registers and jump to `ToStrip`. It shifts the six bytes of `RTL` addresses up by Y bytes, sets up A and the carry flag appropriately, and returns to whomever called the tool.

If the system busy flag needs to be decremented, jump to `ToBusyStrip` instead of `ToStrip`.

How Can I Load My Tool Set From Disk?

One way to load your tool set from disk is to use `InitialLoad` or `InitialLoad2`, supplying a pathname like "9:MyToolset" (prefix 9 is initially set to the directory containing your application; prefix 1 also works, but its length is limited to 64 characters). You can then use `SetTSPtr` to tell the Tool Locator about your tool set, as shown in Appendix A.

Note that `SetTSPtr` calls your `xxxBootInit` function. Even if there is no useful work to be done at BootInit time, you still need to have a `BootInit` function (function number 1) that returns $0000 in the Accumulator and the carry flag cleared..

When you're done with your tool set, call `UserShutdown` on the memory ID returned by `InitialLoad`, so the memory it's using is disposed of or made purgeable. (You can shut it down and allow it to remain in memory in a purgeable state; if you do this, you should try to revive your tool set with `Restart` before you try `InitialLoad` or `InitialLoad2`.)

To allow several applications to share one copy of a user tool set file, you may want to keep your user tool set in the user's `*:System:Tools` folder. To avoid duplicate file names, leave the `ToolXXX` names for System tool sets, and give your user tool set a descriptive name.

If your tool set is not found in the `*:System:Tools` folder, you can then check the `9:` folder. This way users do not need to burden their `*:System:Tools` folders if few of their applications use a particular user tool set or if space on their boot volume is limited.
When your application quits and calls `TLShutDown`, the system disconnects your tool set from the user tool set TPT. If the `UserShutDown` is not followed immediately by the `TLShutDown`, you may wish to use `SetTSPtr` to cleanly remove your tool set from the system (set the tool set pointer so that it points at a zero word).

Note: Because of the way the tool dispatcher transfers control to toolbox functions, a function's entry point must not be at the first byte of a bank ($xx0000). This is normally not an issue, since it's common to put the actual code right after the function pointer table, all in one load segment. Just make sure no function begins

at the first byte of a load segment, and you're safe.

Further Reference

- *Apple IIGS Toolbox Reference*, Volume 2
- *GS/OS Reference*