2010

The Complete Guide to Effective Axe

A Guide for All Things Axe

Drew DeVault
Sir Cmpwn Technologies
6/10/2010

Contents

Introduction

Axe Parser is a powerful and incredible piece of software for the calculator. It has many features and helps TI Basic programmers make programs with the speed and power of assembly in a language similar to TI Basic. It does pose a challenge, however, to convert to Axe. A TI Basic programmer should know several things before using Axe. The introduction covers these items and helps ease new programmers into the world of Axe Parser.

Hexadecimal and Binary

One of the largest issues that an Axe developer faces if they do not know assembly is how to use hexadecimal. Hexadecimal is the way that numbers in base 16 are referred to. Base 16 is a way of counting numbers (Hexa meaning six and deci meaning ten, 6+10=16). Most people are used to using base 10. This means that we use ten digits: 0-9. Once you hit nine, when you add one, you add a digit and start at zero, by going to 10. After nine, you reuse the same digits. In hexadecimal, this set extends to reach F. Digits in hexadecimal move from 0-F, where after nine, the digits A-F are used. In this way, counting from nine and up, hexadecimal represents numbers like so: 9, A, B, C, D, E, F, 10 … 19, 1A, 1B, 1C, 1D, ect.

Binary is similar in that it is a different base. Binary is base two, the second simplest base we can use. Base two is a numbering system that only uses ones and zeros. This is what the computer, and the calculator, knows how to use. Electricity is either flowing or not (two options), so data is represented with zero and one (two options). The number one is 1 in binary, and 1 in decimal. The number two is 2 in decimal, but 10 in binary. After one, we already hit the limit on digits, and have to add a digit.

Hexadecimal and binary are both represented in a common way. In base ten, we commonly do not add extra zeros. To write one, we usually don't write 01. However, in most cases, one would be written as 01 in hexadecimal, because most values are written as bytes. A byte is a number up to FF (or 255 in base ten. Get used to seeing numbers in hex). A word is two bytes, up to 65535, usually represented with four digits, or 0001. The individual digits are called nibbles from time to time.

Binary values are also usually shown with extra digits. They are usually shown with eight digits, where one would be 00000001. With eight binary digits, the highest possible value is 11111111, or 255, or FF. Notice a common theme? They are all represented as bytes. This is because a bytes is the basic variable in any computer system. Axe has their own way of defining hex and binary. Hex can be defined as data with the [] brackets, and as constants with the $_{E}$ token. Binary is defined as constants with a "b" on the end.

Tilemapping

Grayscale

Grayscale in Axe is fairly trivial. It was a bit harder a couple versions back, but since full support has been implemented, it is much easier. I will only discuss how to use 4 level grayscale here. How grayscale on a calculator works is simple: turn on and off the pixels quickly to create the illusion of grayscale. In this case, you can have four shades by having some pixels on 1/3 of the time, and some on 2/3 of the time. This is the method that Axe uses. You are probably familiar with Pt-On, Pt-Off, and Pt-Change. To review, Pt-On will draw a sprite using OR logic, meaning that it will turn on pixels that are off, but won't turn off any pixels, thus making a transparent sprite. Pt-Off will overwrite the sprite, meaning it will clear an 8x8 section of pixels behind the sprite. Pt-Change will use XOR logic, meaning that it will invert the pixels on the screen. Axe provides a few special Pt methods to use grayscale.

In order to use four level grayscale, Axe requires the following command before using any grayscale commands:

Command	Description
SetUpEditor	Must be called once at the start of any program that uses 4 color grayscale.

After this, there are several commands that are used to draw grayscale sprites. The commands are as follows (Quoted from the Axe Documentation):

Command	Description
Pt-On(X,Y,Pic)	The 8x8 sprite that is pointed to is drawn to the buffer at (X,Y). Does not clear the area behind it.
Pt-Off(X,Y,Pic)	The 8x8 sprite that is pointed to is drawn to the buffer at (X,Y) but clears the area behind it first.
Pt-Change(X,Y,Pic)	The 8x8 sprite that is pointed to inverts its pixels on the buffer at (X,Y).
Pt-On(X,Y,Pic)^{r}	The 8x8 sprite that is pointed to is drawn to the back buffer at (X,Y). Does not clear the area behind it.
Pt-Change(X,Y,Pic)^{r}	The 8x8 sprite that is pointed to inverts its pixels on the back buffer at (X,Y).
ClrDraw	Erases the buffer.
ClrDraw^{r}	Erases the back buffer.
DispGraph^{rr}	Draws the buffer on the screen over an evolving pattern of the back buffer. Used for 4 color grayscale. Will not work in full speed mode. This command MUST be initialized with the "SetupEditor" command before it can be used.

The buffer is drawn as light gray, and the back buffer is drawn as dark gray. Sprites drawn on both are drawn as black. Here is an example of a loop to display three sprites as all three colors:

Code **(Source Code: GrayDemo.8xp)**

```
:[FFFFFFFFFFFFFFFF→Pic1  // Black sprite
:SetUpEditor      // Set up grayscale
:While 1          // Loop forever
:ClrDraw          // Clear the buffer
:ClrDrawr         // And the back buffer
:Pt-On(8,0,Pic1   // Draw a light gray sprite at 8, 0
```

```
:Pt-On(16,0,Pic1)r // Draw a dark gray sprite at 16, 0
:Pt-On(24,0,Pic1  // This combined with
:Pt-On(24,0,Pic1)r // this will draw a black sprite at 24, 0
:DispGraphr  // Display the screen, in grayscale
:End
```

Physics

Variable-Length Lists

Optimizations

Floating-Point Math

Data Storage

Graphics

Input

Tricks