TokenIDE v0.10.2 by Merthsoft Creations

Shaun McFall, 2012

shaunm.mcfall@gmail.com

# Note:

This readme is incredibly outdated! Check out the thread on Cemetech for more up-to-date documentation, or join us on EFNet at #cemetech.

<http://www.cemetech.net/forum/viewforum.php?f=69>

The DoorsCS GUI Editor is currently not useable in Linux.

# About:

TokenIDE is an Interactive Development Environment for TI-BASIC. Its goal is to provide BASIC programmers the ability to develop their programs on the computer, and provide tools to make the development process easier. This version has the ability to load and build .8xp programs, with control over how the program is rendered by TokenIDE. All of the symbols used by the calculator are stored in .xml files. By modifying an .xml file, a programmer can change the way the tokens get translated. There are currently three provided .xml files: Tokens1.xml, Tokens2.xml, and AxeTokens.xml. Tokens2.xml is the used by default. Most ambiguities should be resolved in this file. Tokens1.xml was the original file used, based off of SourceCoder's (http://sc.cemetech.net) translation with some modifications. AxeTokens.xml has some changed names that should make Axe development easier. Both Tokens1.xml and Tokens2.xml have built-in support for xLib, Celtic III, PicArc, and the DoorsCS BASIC libs. This should make reading and writing the code for programs that make use of these libs easier.

This is not a complete program yet, and it is still in active development. There will probably be crashes, but there isn't anything terribly unstable about it, you shouldn’t encounter anything that destroys your computer. Please visit http://cemetech.net or http://omnimaga.org (or the IRC channels on EFnet) or email me at shaunm.mcfall@gmail.com with any suggestions or fan mail.

The most recent version can be found at http://merthsoft.com/Tokens.zip

# Using TokenIDE:

When you start the program, you should see what looks like a pretty standard text editor. You can start making your program right there, or choose to open a file. You can open .txt or .8xp files. If you open an .8xp file, the contents will be detokenized to a new tab, but it will be read-only. This is because editing is only supported on .txt files (this makes my life easier). If you want to edit the file, save it as a text file and open that file. Once you’ve got a text file open, you can edit it to your heart’s content. Editing the program from here should be fairly straight forward. If you are unsure what the correct keyword is for a given token, you can try looking in the xml file you are using. When you are done editing your program, go to Build->Compile .8xp (or press F5). This will build the program for you. If you’ve started from scratch, it will prompt you for a program name, otherwise it will give it the name of the file you have open. This will overwrite with no prompting. If you just want to save your work, you can use Ctrl-S to save, or Ctrl-Alt-S to save as (these are also in the menu).

# Library Support:

If you are familiar with the xLib library, you may expect to see something like:

real(1,A,B,2,16,1,2,16,0,0,1

to draw a sprite. With TokenIDE, however, you will see:

DrawSprite(A,B,2,16,1,2,16,0,0,1

These will both tokenize down to the same thing when the program is built, but this should make it easier to see immediately what is going on, and should make the code more readable in general. For more information about the supported libraries, please see:

http://dcs.cemetech.net/index.php?title=Third-Party\_BASIC\_Libraries

and

http://dcs.cemetech.net/index.php?title=DCSB\_Libs

(these are the documents I used as reference when making this). Please note that if you are using a command that has an optional parameter (such as xLib's ClearScreen), you will need to include the comma after the parenthesis, e.g.:

real(0, 1 => ClearScreen(,1

# Comments and Preprocessor:

There are times when you want to add comments in your program, but you don’t want them to show up in the actual on-calc program. If order to do this, you can prefix a line with “//”. This will make the tokenizer ignore that line completely. You can use this to explain what code does, or remove it from the on-calc version. For example, you might have:

Repeat getKey:End

And you might want to explain what it does, so you could put:

// Block program execution until a key is pressed

Repeat getKey:End

There is also a Preprocessor. What this allows you to do is define certain relationships you want to be constant throughout the program. For example, if you’re using the variable K to be your getKey variable throughout the entire program, you can put

#define keyVar K

And now whenever you use keyVar in your program, the tokenizer will replace it with K, so if you do

geyKey->keyVar

If keyVar == 105

Disp “Hello, world!

This will get tokenized to:

getKey →K

If K=105

Disp “Hello, world!

Keep in mind that it will replace ANY occurrence of the word, even if it’s part of another command. This means if you have #define Map [A], it'll replace DrawTileMap with DrawTile[A], which will not give you the expected results. At any point if your program you can undefine as well, which will remove that conversion for the remainder of the program so

#define keyVar K

getKey->keyVar

#undefine keyVar

getKey->keyVar

Will get tokenized to:

getKey →K

getKey →keyVar

You can have a #define or #undefine anywhere in your program. You can also check for defines with conditionals. The possible conditionals are:

#ifdef <variable> – Checks if <variable> has been defined, and includes the code in this block if it has.

#ifndef <variable> – Checks if <variable> isn’t defined, and includes the code in this block if it hasn’t.

#else – Will include the code in this block if the previous ifdef/ifndef has evaluated to false.

#elseifdef <variable> – Will include this block if the previous ifdef/ifndef has evaluated to false, and <variable> is defined.

#elseifndef <variable> – Will include this block if the previous ifdef/ifndef has evaluated to false, and <variable> is not defined.

#endif – Closes this block.

For example, you can use this for debug text:

#define DEBUG

<some code>

#ifdef DEBUG

Disp “The answer was”, K

#endif

Will include the Disp statement if and only if DEBUG has been defined.

# Reference Pane:

The reference pane contains a list of all the functions and commands that get tokenized by TokenIDE. You can select an item to see the syntax for the command, or right click on it to be taken to a webpage that contains more information about it. Double clicking will insert the function at the current location of your editor. This is driven by the XML file, so you can add your own comments as you see fit.

# Tools:

There are currently three tools that should make development easier: a hex sprite editor, a DCS GUI editor (not complete), and an image editor.

## Hex Sprite Editor:

With this tool you can create and insert your own hex sprites for use by various libraries. This should be pretty straight forward. Set the dimensions of the sprite you want, start editing the image, and then hit insert. The hex for the sprite will then be inserted at the cursor in your program. If your program has a hex sprite in it, and you want to edit that, highlight the hex, and then run the hex sprite editor. The program will attempt to get the size of the sprite, and then render it. If the size if not correct, you can adjust the width and height (if Maintain Dim is selected, increasing the height will decrease the width etc.). You can also paste the hex straight into the hex pane and hit "Hex Resize", though this doesn't work quite as well. If Active Hex option is on, the hex in the hex pane will update as you edit the sprite, and the sprite in the sprite pane will update as you edit the hex. Setting the Pixel size zooms the sprite in/out, and the Draw Grid option enables/disables the grid. Under the hex window, there is a binary window that gives you the .dbs for ASM programs. This is read only, but you can copy from it.

## DCS GUI Editor:

This tool is not yet complete. Right now you can place the three container elements, checkboxes, and rectangles. Because this may change quite a bit in the next release, and because it's no near completion, I'm not going to into many details about it. If you want to know more, please visit http://cemetech.net or http://omnimaga.org and post in the respective topics, or visit the IRC channels on EFnet, #cemetech and #omnimaga.

## Image Editor:

This tool lets you open, save, and modify images. You can load .png, .jpg/.jpeg, .bmp, .gif, and .8xi files, and save .8xi files. If you open a non-calc image file, a crop box will open where you can select the region you want to import. Move the selection rectangle until you find the part you want, adjust the tolerance as you see fit, and hit done. The section is converted to black and white by taking the average of the color channels for each pixel, and if that value is greater than the tolerance setting the pixel to white, otherwise setting it to black. You can draw on the picture using the tools provided in the tools pane. Left click to turn a pixel on, right click for off. Once you are done editing you can then save the image to .8xi. There is also the choice to insert the picture as hex or binary (formatted for ASM programs) into the program editor. You can do this as one giant string, or broken up into NxM chunks (as you may want to do it it's a sprite sheet). 8x8 and 16x16 are supported out of the box, but you can set the width and height as you choose, I just can't guarantee that it will work as expected (this feature is still being tested).

# TokenIDE.ini File:

The .ini file contains a lot of configuration for TokenIDE including font, default tokens files, and external tool support.

## Font:

Use this section to control the font that is used within TokenIDE. This can also be changed from within TokenIDE from Tools->Options.

## TokenIDE:

Use this section to control the default XML file when creating a new program or opening a program from a text file or binary file. This can also be changed from within TokenIDE from Tools->Options. This can also be changed from within TokenIDE from Tools->Options.

## Extensions:

Use this section to control the XML file that is used when opening specific calculator types.

## Tools:

Use this section to add support for specific tools. The syntax for a line is:

tool#=[Delimiter][Tool name][Delimiter][Tool path] [Delimiter] [Parameters]

[Delimiter] can be any character, but has to be the first character of the line. There are predefined tokens that will be replaced from within TokenIDE:

%file% - Replaced with the path to the currently edited file.

%files% - Replaced with the path to all open files.

Example:

This will send all open files to Wabbitemu, assuming it’s installed at C:\Wabbit:

tool2=^To Wabbit^"C:\Wabbit\wabbitemu.exe"^%files%

# Acknowledgements:

Many thanks go out to Christopher Mitchell, Weregoose, Ben Ryves, Sonlen, Muffinator, Kévin Ouellet, Runer112, and everyone else at Cemetech and Omnimaga.

# Mumbo-Jumbo:

This program is released with no warranty, and is completely closed-source. You may not release an altered version without the permission of Shaun McFall, and may no part of this software be otherwise bundled as part of any other package without explicit permission.