
Celtic CE Documentation

Release Beta 1.1

RoccoLox Programs, Tiny_Hacker

Nov 29, 2022

OVERVIEW

1	Overview	1
2	Documentation	3
2.1	Installation	3
2.2	General Syntax	4
2.3	Error Codes	6
2.4	TI-84 Plus CE Hex Codes	6
2.5	Colors	9
3	Function Syntax	11
3.1	Celtic 2 CSE Functions	11
3.2	Doors CE 9 Functions	16
3.3	Graphics Functions	17
3.4	OS Utility Functions	21
3.5	Celtic III Functions	24

OVERVIEW

Celtic CE is a BASIC library written in eZ80 ASM for the TI-84 Plus CE/Premium CE family of calculators. This documentation aims to help users understand how to use Celtic in their programs. The documentation section contains information about Celtic CE in general. It is advised to read this before using Celtic in your programs if you do not have experience with BASIC libraries. The function syntax section contains detailed syntax for each function, along with possible error codes that might not be obvious.

A pdf form of the documentation can be downloaded [here](#).

2.1 Installation

2.1.1 Downloading the Installer

- Download the latest release of Celtic CE from the [GitHub releases](#) page.
- Transfer CelticCE.8xp to your calculator using **TI-Connect CE** or **TiLP**.

2.1.2 Installing the App

- Open the programs menu and select **CELTICCE**.
- Run **CELTICCE** and follow through the installation process.

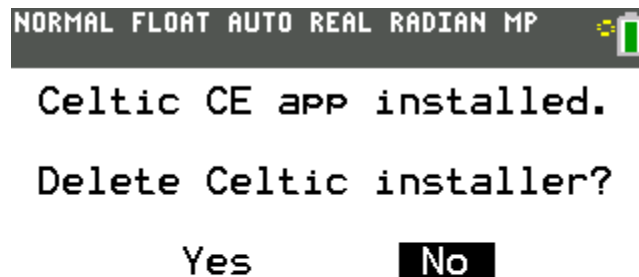


Fig. 1: The Celtic installer menu.

Note: If you delete the installer, you'll need to send it to the calculator again, should the app ever get deleted. However, you can still uninstall and re-install the Celtic CE library using the app. The program is only used for installing the app onto the calculator.

- Open the apps menu and select **CelticCE**.
- Press 1 to install. You can now exit the app by pressing 3 or **clear**.

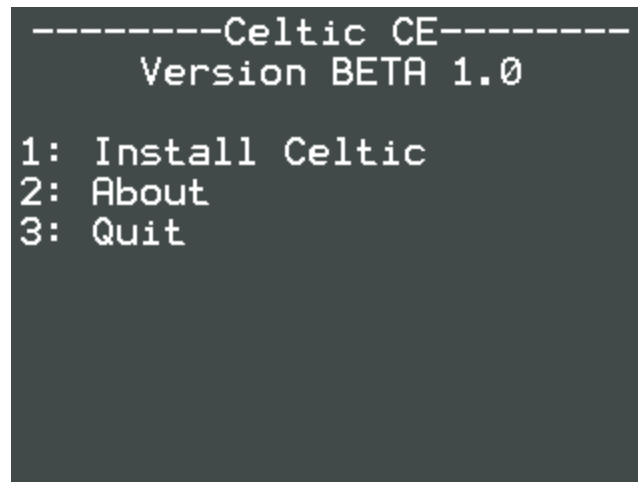


Fig. 2: The Celtic app.

Note: If you install Celtic CE when another parser hook (**ASMHOOK**, for example) is installed, it will chain with it in order to preserve it. This means that both hooks will be able to run in tandem. However, if Celtic gets uninstalled, you will need to re-install the second hook again.

2.2 General Syntax

2.2.1 Overview

Celtic uses a parser hook to search for a token in your program and run its ASM code if it finds it. When it encounters the `det(` token, Celtic will detect which function you are calling and any arguments with it. The first argument after the `det(` token tells Celtic which function you wish to call. If you have entered a valid argument, Celtic will tell you what function the argument is referencing in the status bar of the program editor when your cursor is over it.

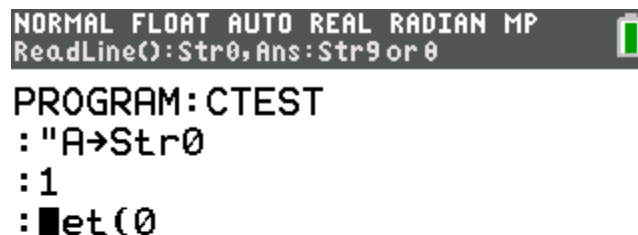


Fig. 3: Celtic's function preview feature.

The function preview follows a general syntax: “CommandName(Arguments): Input Vars (if any): Output Vars (if any)”. As you can see in the example above, ReadLine is listed with no arguments, Str0 and Ans as the input variables, and Str9 or theta as the output variables.

Note: In the event that you pass invalid arguments to Celtic, it will return an error. All errors are returned in Str9. See the [Error Codes](#) page for more information.

2.2.2 Check if Celtic is Installed

If your program uses Celtic, it is recommended that it makes sure Celtic is installed when ran. To check if Celtic is installed, it is recommended you put something like this at the beginning of your program:

```

::DCS
:"Icon data...
:If 90>det([[20
:Then
:Disp "Get Celtic CE to run this:", "bit.ly/CelticCE
:Return
:End

```

det([[20]]) will equal 90 if Celtic CE is installed. If the program aims to be compatible with Celtic 2 CSE, you may wish to refer to the [list of version codes](#) on the DCS Wiki as well.

2.2.3 Argument Types

Arguments are passed to Celtic in three different ways. They can be passed in the det(function, in the Ans variable, or in a string. The method of passing arguments will differ, depending on the function and the type of argument that is being passed. The specific syntax for each function is listed in the documentation.

Most often, numerical arguments are passed in the det(function. Variable names are put in Str0, and strings arguments are put in Str9. The exact usage will vary depending on the function. Celtic only accepts positive real integers as arguments. If a decimal number is passed, Celtic will only take the integer part of it.

When using Str0 as a program name, simply store the name of the program into Str0. When storing an AppVar, begin the string with the rowSwap(token. For example:

```

:"FOO" -> Str0           //program F00
:"rowSwap(F00" -> Str0   //AppVar F00

```

If a string is used, it must be in the RAM when the function is called.

2.2.4 Returns

Depending on the function, Celtic will return a value after it runs. For example, the SpecialChars function fills Str9 with certain special characters.

String returns (including errors) are most often in Str9, while numerical returns are in the theta variable. All functions will return the initial contents of Ans (in order to preserve it).

2.3 Error Codes

2.3.1 Overview

Celtic does its best to prevent errors. However, if something goes wrong with running a function, Celtic will return an error code. This can be from lack of memory, an incorrect argument, or something else. Keep in mind that Celtic might not always detect the exact error, or realize that there is one in the first place. You can use the error codes to help with the debugging process.

2.3.2 Documentation

Error Code	Description
..P:IS:FN	A program already exists, and Celtic will not overwrite it.
..NUMSTNG	The specified line is past end of file. Used when getting the amount of lines in a program.
..L:NT:FN	The specified line in the program does not exist.
..S:NT:FN	The specified string was not found.
..S:FLASH	The specified string is archived.
..S:NT:ST	The input is not a string, but Celtic expects a string.
..NT:EN:M	There is not enough memory to complete the requested action.
..P:NT:FN	The specified program or appvar was not found.
..PGM:ARC	The specified program or appvar is archived, so it cannot be modified.
..NULLSTR	The specified line exists, but is empty.
..NT:REAL	The input was not a real number, but Celtic expects a real number.
..INVAL:A	An invalid argument was entered. This error code could refer to various issues.
..INVAL:S	An invalid string was entered.
..2:M:ARG	More arguments were entered than Celtic is able to handle.
..NT:A:LS	The argument was not a list, but Celtic expects a list.
..E:NT:FN	The entry was not found in the list specified.
..SUPPORT	Whatever happened was not supported by Celtic.

2.4 TI-84 Plus CE Hex Codes

2.4.1 Overview

This page contains a list of useful hex codes to be used with ExecHex. To use them, follow the ExecHex documentation with the preferred hex string. Some of these may perform similar utilities to pre-existing Celtic functions. A good number of these were taken/adapted from the TI-BASIC Developer [84 Plus CE Hexcodes](#) page.

2.4.2 Documentation

Turn Off LCD

Turns off the calculator's LCD:

"3E01321900E3C9"

Turn On LCD

Turns on the calculator's LCD:

```
"3E09321900E3C9"
```

Set LCD Brightness

Set the LCD brightness to a value between 0 and 255 stored in Ans, 255 is low, 0 is high:

```
"CD500F02CD700F02322400F6C9"
```

Decrease Brightness

Decrease the LCD brightness by 1:

```
"3A2400F63C322400F6C9"
```

Increase Brightness

Increase the LCD brightness by 1:

```
"3A2400F63D322400F6C9"
```

Invert LCD (High contrast mode/Dark mode)

Inverts the colors of the LCD. Also sometimes referred to as “High contrast mode” or “Dark mode”:

```
"211808F874364436216C3601C9"
```

Credit: MateoConLechuga

Toggle Program mode

When used in a program, it allows you to use Archive and UnArchive on other programs:

```
"FD7E08EE02FD7708C9"
```

Warning: Make sure to switch back to “program mode” when you’re done by running the program again.

Quick Key

Loads the most recent keycode into Ans:

```
"3A8705D0CD080B02CD300F02C9"
```

Text Inverse

This will switch from normal text mode to inverse (white text on black background) and vice versa:

```
"FD7E05EE08FD7705C9"
```

Enable Lowercase

Enables lowercase letters in TI-OS:

```
"FDCB24DEC9"
```

Disable Lowercase

Disables lowercase letters in TI-OS (default):

```
"FDCB249EC9"
```

Toggle Lowercase

Toggles lowercase letters on/off in TI-OS:

```
"FD7E24EE08FD7724C9"
```

Clear LCD

Clears the LCD:

```
"CD101A02C9"
```

Clear LCD and Redraw Status Bar

Same as Clear LCD, but redraws the Status Bar as well:

```
"CD101A02CD3C1A02C9"
```

Fill Screen with White

Fills the screen with white:

```
"210000D436FFE5D11301FF5702EDB0C9"
```

Fill Screen with Black

Fills the screen with black:

```
"210000D43600E5D11301FF5702EDB0C9"
```

Fill Screen with xLIBC Color Value in Ans

Fills the screen with an xLIBC Color in Ans:

```
"CD500F02CD700F02320000D4210000D4E5D11301FF5702EDB0C9"
```

Run Indicator Off

Turns off the run indicator:

```
"CD480802C9"
```

Run Indicator On

Turns on the run indicator:

```
"CD440802C9"
```

Toggle Run Indicator

Toggles the run indicator on/off:

```
"FD7E12EE01FD7712C9"
```

Disable APD

Disables Automatic Power Down (APD):

```
"CD341102C9"
```

Enable APD

Enables Automatic Power Down:

```
"CD381102C9"
```

Turn Off Cursor

This is harmless, but it stops displaying that blinking cursor :D Just press [2nd][MODE] to put it back to normal. What, jokes are allowed, right?

– TI-BASIC Developer

```
"FDCB0CE6C9"
```

Turn On Cursor

Turns on the cursor:

```
"FDCB0CA6C9"
```

Draw TI Logo

This is a strange function that draws the TI Logo. (Yes, there is a built in ASM call to do that) While there is no real reason you would probably want to do this, it's still interesting:

```
"CD001B02C9"
```

Trigger Error

This code will cause an error to be thrown, based on the value in Ans. The values correspond to the ones found [here](#). Use your browser's search tool to look for "E_Overflow" which is Error 1. If you are using a Windows or Chromebook, pressing Control+F will bring up the search tool. On a Mac, pressing Command+F will bring up the search tool. By default, there is no Goto option unless you add 128 to the values. So for example, for a divide by zero error with a goto option, use 130 as the error code.

– TI-BASIC Developer

```
"CD500F02CD700F02CD900702C9"
```

2.5 Colors

2.5.1 Overview

Some Celtic functions allow RGB 565 colors. A list of pre-defined colors can be found below. You can also use the [1555 color picker](#) in 565 mode.

2.5.2 Colors

Refer to the DCS wiki for a [list of pre-defined colors](#).

FUNCTION SYNTAX

3.1 Celtic 2 CSE Functions

3.1.1 Overview

These functions are the same as those included in Doors CSE 8 for the TI-84 Plus CSE, unless noted otherwise. Most documentation is from the [DCS Wiki](#).

3.1.2 Documentation

ReadLine: det(0), Str0 = variable name, Ans = line number

Reads a line from a program or AppVar. If Ans (line number) equals 0, then Theta will be overwritten with the number of lines in the program being read. Otherwise, Ans refers to the line being read.

Warning: If you attempt to read the line of an assembly program, there is a risk of a reset. If Celtic passes an invalid token to Str9, it could cause a RAM clear.

Parameters:

- Str0: Name of program to read from.
- Ans: Line number to read from, begins at 1.

Returns:

- Str9: Contents of the line read.
- Theta: Number of lines if Ans was 0.

Errors:

- . . NULLSTR if the line is empty.

ReplaceLine: det(1), Str0 = variable name, Ans = line number, Str9 = replacement

Replaces (overwrites) a line in a program or AppVar. Ans refers to the line to replace.

Parameters:

- Str0: Name of file to read from.
- Ans: Line number to replace, begins at 1.

- Str9: Contents to replace the line with.

Returns:

- Str9: Intact if no error occurred; otherwise, contains an error code.

Errors:

- ..PGM:ARC if the file is archived.
-

InsertLine: det(2), Str0 = variable name, Ans = line number, Str9 = contents

Inserts a line into a program or AppVar. Ans refers to the line number to write to.

Parameters:

- Str0: Name of file to write to.
- Ans: Line number to write to, begins at 1.
- Str9: Material to insert into a program. The line that was occupied is shifted down one line and this string is inserted into the resulting location.

Returns:

- Str9: Intact if no error occurred; otherwise, contains an error code.

Errors:

- ..PGM:ARC if the file is archived.
 - ..NT:EN:M if there is not enough memory to complete the action.
-

Note: If your usage of InsertLine results in the program exceeding 65535 bytes (the maximum size of a file), it will result in loss of memory. Celtic does not check if you exceed this filesize, as there should be no reason anyone would do this in the first place.

SpecialChars: det(3)

Stores the -> and " characters into Str9.

Returns:

- Str9: -> and ", respectively. You can use substrings to extract them. There are also 7 more characters in Str9, which are junk.
-

CreateVar: det(4), Str0 = variable name

Create a program or AppVar given a name.

Parameters:

- Str0: Name of program or AppVar to create.

Alternative method for appvars: det(4, HEADER), Str0 = variable name

- HEADER: whether or not to include a header which allows [CEaShell](#) to edit the appvar. This extra argument is optional. 1 to include the header, and 0 to not.

Returns:

- Str9: Intact if no error occurred; otherwise, contains an error code.
-

- Str0: Intact with program's name to be created.

Errors:

- ..P:IS:FN if the program already exists.
-

ArcUnarcVar: det(5), Str0 = variable name

Archive/unarchive a program or AppVar given a name.

Parameters:

- Str0: Name of program or AppVar to move between Archive and RAM.

Returns:

- Moves a program or AppVar into RAM if it was in Archive, or into Archive if it was in RAM.
-

DeleteVar: det(6), Str0 = variable name

Delete a program variable or an AppVar given a name.

Parameters:

- Str0: Name of program or AppVar to delete.

Returns:

- The indicated program or AppVar is deleted.
-

DeleteLine: det(7), Str0 = variable name, Ans = line number

Deletes a line from a program or AppVar. Ans is the line to delete.

Parameters:

- Str0: Name of program or AppVar to delete from.
- Ans: Line number to delete from, begins at 1.

Returns:

- Str9: Intact if no error occurred; otherwise, contains an error code.
-

VarStatus: det(8), Str0 = variable name

Output status string describing a program or AppVar's current state, including size, visibility, and more.

Parameters:

- Str0: Name of program or AppVar to examine.

Returns:

- **Str9: Contains a 9 byte output code.**
 - 1st character: A = Archived, R = RAM
 - 2nd character: V = Visible, H = Hidden
 - 3rd character: L = Locked, W = Writable
 - 4th character: _ (Space character)
 - 5th - 9th character: Size, in bytes

- Example: AVL 01337 = Archived, visible, locked, 1337 bytes.
-

BufSprite: det(9, width, X, Y), Str9 = sprite data

Draws indexed (palette-based) sprite onto the LCD and into the graph buffer. Copies the contents of the graph buffer under the sprite back into Str9, so that you can “erase” the sprite back to the original background. Good for moving player characters, cursors, and the like. Interacts politely with Pic variables and OS drawing commands like Line(), Circle(), Text(), and so on. If you want to draw a lot of different sprites to the screen and won’t need to erase them back to the background, then use BufSpriteSelect instead.

Parameters:

- Str9: Sprite data as ASCII hex, one nibble per byte. The digits 1-F are valid colors (1 = blue, 2 = red, 3 = black, etc), while G will cause the routine to skip to the next line. 0 is normal transparency, and lets the background show through. H is a special kind of transparency that erases back to transparency instead of leaving the background color intact.
- X: X coordinate to the top-left corner of the sprite.
- Y: Y coordinate to the top-left corner of the sprite.
- width: Sprite width (height is computed).

Returns:

- Str9: Same length as input, contains the previous contents of the graph buffer where the sprite was drawn. You can call det(9...) again without changing Str9 to effectively undo the first sprite draw.

Errors:

- ..INVALID:S if the string contains invalid characters.
-

BufSpriteSelect: det(10, width, X, Y, start, length), Str9 = sprite data

Draws indexed (palette-based) sprite onto the LCD and into the graph buffer. Good for drawing tilemaps, backgrounds, and other sprites that you won’t want to individually erase. If you want to be able to erase the sprite drawn and restore the background, you should consider BufSprite instead. This routine takes an offset into Str9 and a sprite length as arguments, so that you can pack multiple sprites of different lengths into Str9.

Parameters:

- Str9: Sprite data as ASCII hex, one nibble per byte. The digits 1-F are valid colors (1 = blue, 2 = red, 3 = black, etc), while G will cause the routine to skip to the next line. 0 is normal transparency, and lets the background show through. H is a special kind of transparency that erases back to transparency instead of leaving the background color intact.
- X: X coordinate to the top-left corner of the sprite.
- Y: Y coordinate to the top-left corner of the sprite.
- width: Sprite width (height is computed).
- start: Offset into Str9 of the start of pixel data, begins at 0.
- length: Length of sprite data in characters.

Returns:

- Sprite drawn to LCD and stored to graph buffer.

Errors:

- ..INVALID:S if the string contains invalid characters.

ExecArcPrgm: det(11, function, temp_prog_number), Ans = program name

Copies a program to the XTEMP program of the specified temp_prog_number. Ans is the name of the program to copy. function refers to the behavior of the ExecArcPrgm command, as seen in the table below:

Code	Function
0	Copies the program in Ans to the XTEMP program specified.
1	Deletes the XTEMP program with the specified number.
2	Deletes all XTEMP programs.

Parameters:

- function: The requested behavior of the function. Can be 0, 1, or 2.
- temp_prog_number: The number of the XTEMP program to create/delete.
- Ans: Name of program to copy from.

Returns:

- Completes the specified function.

Errors:

- ..NT:EN:M if there is not enough memory to complete the action.
-

DispColor: det(12, FG_LO, FG_HI, BG_LO, BG_HI)

Changes the foreground and background color for Output(), Disp, and Pause to arbitrary 16-bit colors, or disables this feature. Due to technical limitations, the foreground and background for Text() cannot be changed to arbitrary colors. To disable this mode, you should call det(12, 300) before exiting your program.

Parameters:

- FG_LO: low byte of foreground color.
- FG_HI: high byte of foreground color.
- BG_LO: low byte of background color.
- BG_HI: high byte of background color.

Alternative method: det(12, FG_OS, BG_OS)

- FG_OS: Foreground color from TI-OS Colors menu, like RED or BLUE or NAVY.
- BG_OS: Background color from TI-OS Colors menu, like RED or BLUE or NAVY.

Colors:

- A list of colors can be found [here](#).

Returns:

- See description.

3.2 Doors CE 9 Functions

3.2.1 Overview

These functions are roughly based/related to some of the functions made for Doors CE 9.

3.2.2 Documentation

DispText: `det(13, LargeFont, FG_LO, FG_HI, BG_LO, BG_HI, X, Y), Str9 = text`

Displays colored text from Str9 at X and Y on the screen, using the OS large or small font.

Warning: You can only use a maximum of 128 characters in Str9 at a time with this command. However, this should be plenty, since the text does not wrap.

Parameters:

- **LargeFont** = whether to use OS large or small font. 0 means to use the OS small font, and 1 means to use the large font.
- **FG_LO**: low byte of foreground color.
- **FG_HI**: high byte of foreground color.
- **BG_LO**: low byte of background color.
- **BG_HI**: high byte of background color.
- **X**: X location to display the text, starting from the top-left corner.
- **Y**: Y location to display the text, starting from the top-left corner.
- **Str9**: Text to display.

Alternative method: `det(13, LargeFont, FG_OS, BG_OS, X, Y)`

- **FG_OS**: Foreground color from TI-OS Colors menu, like RED or BLUE or NAVY.
- **BG_OS**: Background color from TI-OS Colors menu, like RED or BLUE or NAVY.

Colors:

- A list of colors can be found [here](#).

Returns:

- Displays the specified text.

ExecHex: `det(14), Ans = hex code`

Executes the string of ASCII-encoded hexadecimal in Ans. Although a C9 (ret) at the end of your hex string is highly encouraged, Celtic will automatically put one at the end for safety regardless. For a list of useful hex codes, refer to [this page](#).

Warning: Ans must be under (not including) 8192 characters. It also must be an even number of characters.

Parameters:

- Ans: hex code to execute.

Returns:

- Runs the specified hex code.

Errors:

- `..INVAL:S` if there is an invalid hex digit or an odd number of characters in the string.
-

TextRect: det(15, LOW, HIGH, X, Y, WIDTH, HEIGHT)

Draw a filled, colored rectangle on the screen. This command can also be used to draw an individual pixel by setting the width and height to 1, or a line by setting either the width or height to 1.

Parameters:

- LOW: low byte of color.
- HIGH: high byte of color.
- X: X location to draw the rectangle, beginning at the top-left corner.
- Y: Y location to draw the rectangle, beginning at the top-left corner.
- WIDTH: Width of rectangle.
- HEIGHT: Height of rectangle.

Alternative method: det(15, OS_COLOR, X, Y, WIDTH, HEIGHT)

- OS_COLOR: Color from TI-OS Colors menu, like RED or BLUE or NAVY.

Note: If you use the alternative method and use 0 for OS_COLOR, it will invert the section of the screen covered by the rectangle instead of drawing a color. This can be useful for blinking cursors, etc.

Colors:

- A list of colors can be found [here](#).

Returns:

- Draws the colored rectangle.

3.3 Graphics Functions

3.3.1 Overview

These functions are related to graphical operations. This is not the only section containing graphical operations, as some are better categorized in other sections.

3.3.2 Documentation

FillScreen: det(16, LOW, HIGH)

This function fills the screen with the color specified. It is faster than using TextRect to draw a rectangle over the entire screen.

Parameters:

- LOW: Low byte of color.
- HIGH: High byte of color.

Alternative method: det(16, OS_COLOR)

- OS_COLOR: Color from TI-OS Colors menu, like RED or BLUE or NAVY.

Colors:

- A list of colors can be found [here](#).

Returns:

- Fills the screen with the specified color.
-

DrawLine: det(17, LOW, HIGH, X1, Y1, X2, Y2)

Draws a line of the specified color from (X1, Y1) to (X2, Y2).

Parameters:

- LOW: Low byte of color.
- HIGH: High byte of color.
- X1: X location to begin drawing the line from, beginning from the top-left corner of the screen. X has a range of 0 - 319.
- Y1: Y location to begin drawing the line from, beginning from the top-left corner of the screen. Y has a range of 0 - 239
- X2: X location to finish drawing the line at, beginning from the top-left corner of the screen.
- Y2: Y location to finish drawing the line at, beginning from the top-left corner of the screen.

Alternative method: det(17, OS_COLOR, X1, Y1, X2, Y2)

- OS_COLOR: Color from TI-OS Colors menu, like RED or BLUE or NAVY.

Colors:

- A list of colors can be found [here](#).

Returns:

- Draws a line of the specified color from (X1, Y1) to (X2, Y2).
-

SetPixel: det(18, LOW, HIGH, X, Y)

Sets the pixel located at (X, Y) to the color specified.

Parameters:

- LOW: Low byte of color.
- HIGH: High byte of color.

- X: X location of the pixel to set, beginning from the top-left corner of the screen.
- Y: Y location of the pixel to set, beginning from the top-left corner of the screen.

Alternative method: `det(17, OS_COLOR, X, Y)`

- OS_COLOR: Color from TI-OS Colors menu, like RED or BLUE or NAVY.

Note: If you use the alternative method and use 0 for OS_COLOR, it will invert the specified pixel instead of drawing a color.

Colors:

- A list of colors can be found [here](#).

Returns:

- Sets the pixel at (X, Y) to the color specified, or inverts it if using the alternative method with OS_COLOR being 0.
-

GetPixel: `det(19, X, Y)`

Returns a LOW and HIGH byte representing the color of the pixel at (X, Y) in Ans and Theta respectively.

Parameters:

- X: X location of the pixel to check, beginning from the top-left corner of the screen.
- Y: Y location of the pixel to check, beginning from the top-left corner of the screen.

Returns:

- Ans: LOW byte of the color of the pixel checked
 - Theta: HIGH byte of the color of the pixel checked
-

PixelTestColor: `det(20, ROW, COLUMN)`

This function works just like OS function `pxl-Test()` does, however, it will return 0 if no pixel is present and the OS color of the pixel if one is present. This only applies to the graph screen, like `pxl-Test()`.

Parameters:

- ROW: Row of the graphscreen that contains the pixel to test.
- COLUMN: Column of the graphscreen that contains the pixel to test.

Tip: The arguments and functionality of this are identical to `pxl-Test()`, other than the fact that this returns the color of the pixel if one is present.

Returns:

- Theta: 0 if no pixel was present, otherwise will contain the OS color of the pixel tested.
-

PutSprite: det(21, X, Y, WIDTH, HEIGHT), Str9 = sprite data

This function draws a sprite at (X, Y) with a width of WIDTH and a height of HEIGHT. It is designed to be fast, and so it does not have as much error checking, meaning that it will display a sprite of the given width and height regardless of the length of the given sprite data. The sprite data is made up of hex values referring to xLIBC colors, which can be found [here](#). The data is stored left to right and top to bottom. For example, take a sprite that looks like this:



Fig. 1: A sample sprite.

We'll convert it into a matrix, where each pixel is replaced with the hex equivalent of its xLIBC color:

```
[FF, FF, 00, 00, 00, 00, FF, FF
FF, 00, E6, E6, E6, E6, 00, FF
00, E6, 00, E6, E6, 00, E6, 00
00, E6, 00, E5, E5, 00, E6, 00
00, E5, E5, E5, E5, E5, E5, 00
00, E5, E5, 00, 00, E5, E5, 00
FF, 00, E5, E5, E5, E5, 00, FF
FF, FF, 00, 00, 00, 00, FF, FF]
```

Then, to make it a string, we'll take remove the newlines and commas, like this:

```
"FFFF00000000FFFFF00E6E6E6E600FF00E600E6E600E60000E600E5E500E600
00E5E5E5E5E50000E5E500FF00E5E5E500FFFFF000000000FFFF" -> Str9
```

Parameters:

- X: X location to draw the sprite, beginning at the top-left corner of the screen.
- Y: Y location to draw the sprite, beginning at the top-left corner of the screen.
- WIDTH: Width of the sprite in pixels.
- HEIGHT: Height of the sprite in pixels.
- Str9: Sprite data, as explained above.

Colors:

- Uses hex codes referring to the xLIBC colors. A good resource for the xLIBC palette can be found [here](#).

Returns:

- Draws a sprite of the specified width and height at (X, Y).

Warning: Keep in mind that the function will not check if your string is long enough for the provided width and height. If your string is an incorrect size, it will still draw a sprite of the specified width and height, though parts of the drawn sprite could be garbage.

3.4 OS Utility Functions

3.4.1 Overview

These functions perform various actions related to the OS and its various features.

3.4.2 Documentation

GetMode: det(22, MODE)

Checks a specified mode, from 0 to 10. A table containing the information on the modes and possible outcomes is below.

Input	Mode	Possible Returns
0	Mathprint	0 if not enabled, 1 if enabled.
1	Notation	0 for Normal, 1 for Scientific, 2 for Engineering.
2	Trig Mode	0 for Radian, 1 for Degree.
3	Graph Mode	0 for Function, 1 for Parametric, 2 for Polar, 3 for Sequence.
4	Graph Equations	0 for Sequential, 1 for Simultaneous.
5	Numeric Format	0 for Real, 1 for a + bi mode, 2 for re^(Thetai). Theta is the theta token.
6	Window Mode	0 for Full, 1 for Horizontal, 2 for Graph-Table.
7	Coordinate Mode	0 for Rectangular coordinates, 1 for Polar coordinates
8	Grid Mode	0 if the grid is off, 1 if the grid is on.
9	Axis Mode	0 if the axis is off, 1 if it is on.
10	Axis Labels	0 if the axis labels are off, 1 if they are on.

Parameters:

- **MODE:** Which mode to check. Refer to the table above. Must be between 0 and 10.

Returns:

- Varies based on the specified mode. See table above.

RenameVar: det(23), Str0 = variable to rename, Str9 = new name

Renames the variable specified in Str0 with a new name specified in Str9. For appvars, the name in Str0 must be preceded with the rowSwap(token, however, the new name does not need the rowSwap(token. Renaming a program will result in the program being locked.

Parameters:

- **Str0:** The name of the variable you wish to rename. If it is an appvar, it must be preceded by the rowSwap(token.

- **Str9**: The new name that you wish to rename the specified file to. It does not need to be preceded by the `rowSwap(` token, regardless of whether it is a program or appvar.

Returns:

- Renames the variable specified in **Str0** with the new name specified in **Str9**.

Errors:

- `..NT:EN:M` depending on the amount of remaining memory and the size of the variable being renamed.
-

LockPrgm: det(24), Str0 = variable to lock.

Toggles the locked attribute of the program referenced by **Str0**.

Warning: If you lock/unlock an archived program, Celtic un-archives it when running the function and then re-archives it when the function is complete. This means that it could result in a garbage collect.

Parameters:

- **Str0**: The name of the program to toggle the locked attribute of.

Returns:

- Toggles whether or not the specified program is locked.

Errors:

- `..SUPPORT` if you attempt to use this function on an appvar.
-

HidePrgm: det(25), Str0 = variable to hide.

Toggles the hidden attribute of the program referenced by **Str0**.

Warning: If you hide/unhide an archived program, Celtic un-archives it when running the function and then re-archives it when the function is complete. This means that it could result in a garbage collect.

Parameters:

- **Str0**: The name of the program to toggle the hidden attribute of.

Returns:

- Toggles whether or not the specified program is hidden.

Errors:

- `..SUPPORT` if you attempt to use this function on an appvar.
-

PrgmToStr: det(26, STRING_NO), Str0 = variable to read

Copies the contents of a file specified in **Str0** to the string specified by **STRING_NO**. If you wish to read the contents of an appvar, you must precede the name with the `rowSwap(` token in **Str0**.

Parameters:

- **STRING_NO**: The number of the string to copy to. Can be from 0 to 9. 0 means **Str0**, 1 means **Str1** and so on.
- **Str0**: Name of the variable to copy. The name must be preceded by the **rowSwap(** token if you wish to read an appvar.

Returns:

- The contents of the specified variable in the string specified by **STRING_NO**.

Errors:

- **..NT:EN:M** if there is not enough memory to create the string with the contents of the specified variable.
-

GetPrgmType: det(27), Str0 = program to check

Gets the type of program specified in **Str0**. This is not the OS type, it is the actual program type (C, ASM, etc). A table with the return codes and filetypes they signify is below.

Code	Filetype
0	eZ80 Assembly
1	C
2	TI-BASIC
3	ICE
4	ICE Source

Parameters:

- **Str0**: Name of the program to check. It cannot be an appvar.

Returns:

- **Theta**: Contains the number referencing the filetype. See the table above.

Errors:

- **..INVAL:S** if you attempt to use this function on an appvar.
-

GetBatteryStatus: det(28)

Gets the current status of the battery, as a number between 0 and 4, 0 being no charge and 4 being fully charged. If the battery is charging, 10 will be added. For example, a battery that is partially charged and also actively charging would return 12.

Returns:

- **Theta**: Current status of the battery.
-

SetBrightness: det(29, BRIGHTNESS)

Sets the LCD to the specified **BRIGHTNESS**. The brightness can be between 1 and 255, with 1 being the brightest and 255 being the darkest. If the brightness is set to 0, it will instead return the current brightness of the screen.

Note: The brightness will not persist after the calculator is turned off. Instead, it will go back to what it was previously.

Parameters:

- **BRIGHTNESS:** The level of brightness to set the screen to, between 1 and 255. 0 will instead return the current level of brightness.

Returns:

- **Theta:** If you attempt to set the brightness to 0, **Theta** will contain the current brightness.
- If **BRIGHTNESS** is between 1 and 255 (Not 0), it will instead set the screen to the specified brightness, with 1 being the lightest and 255 the darkest.

3.5 Celtic III Functions

3.5.1 Overview

Some (not all) of the functions from Celtic III are included in Celtic CE.

3.5.2 Documentation

GetListElem: det(30, list_element), Ans = list name

Gets a value at `list_element` in the list specified. For example, with **Ans** equal to “`_LFOO`” (Where `_L` is the list token found in `List → OPS`). This is useful if the list being accessed is archived, for example.

Parameters:

- **list_element:** Element of the list to access, beginning at 1. Accessing 0 will return the dimension of the list.
- **Ans:** Name of the list to access. Begins with the `_L` token found in the `List → OPS`. (2nd + stat + left arrow + alpha + apps).

Returns:

- **Theta:** The number at the element of the list accessed, or the dimension of the list if `list_element` was 0.

Errors:

- `..NT:A:LS` if the user did not specify a valid list.
- `..E:NT:FN` if the entry was not found in the list specified.

GetArgType: det(31), Ans = argument to check

Outputs a real number depicting the type of argument in **Ans**.

Parameters:

- **Ans:** Argument to check the type of

Returns:

- **Theta:** The number corresponding to the argument’s type. A table with the possible types is below.

Number	Type
0	Real
1	List
2	Matrix
4	String
12	Complex
13	Cpx List

ChkStats: det(32, function)

This is a multi-purpose command used to read various system statuses. The output will vary based on the specified function. A table with the possible functions and their resulting outputs is below.

Function	Output
0	Total free RAM (In Str9)
1	Total free ROM (Also in Str9)
2	Bootcode (Also in Str9)
3	OS Version (Also in Str9)
4	Hardware version: 0 for 84 Plus CE, 1 for 83 Premium CE (Returns in Theta)

Parameters:

- **function:** Function to complete

Returns:

- Varies based on input
-

FindProg: det(33, type), Ans = search string

This does not search for file names, instead it searches for file contents. The search string checks for the beginning contents of a program. Type refers to the type of file to search, with 0 being programs and 1 being appvars. The function will return a string containing the names of the files containing the search phrase, with each name separated by a space. For example, to look for all the programs beginning with “:DCS”, your code would look something like this:

```
" :DCS
det(33, 0)
```

Parameters:

- **type:** Type of file to search. 0 for programs, 1 for appvars.
- **Ans:** Content string to search for. If Ans does not contain a string, it will return a complete list of all the files of the specified type.

Returns:

- **Str9:** Contains a list with the names of files containing the search string, separated by spaces. If Ans was not a string, it returns a list of all files of the specified type.

Errors:

- **..P:NT:FN** if no files are found containing the specified search string.