Celtic CE Documentation

Release Beta 1.3

RoccoLox Programs, TIny_Hacker

Jun 01, 2023

OVERVIEW

CHAPTER

ONE

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.

CHAPTER

TWO

DOCUMENTATION

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.

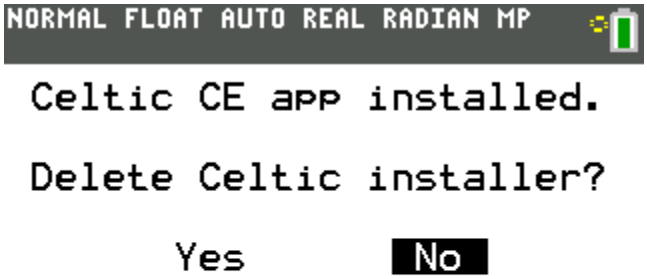

Fig. 1: The Celtic installer menu.

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.

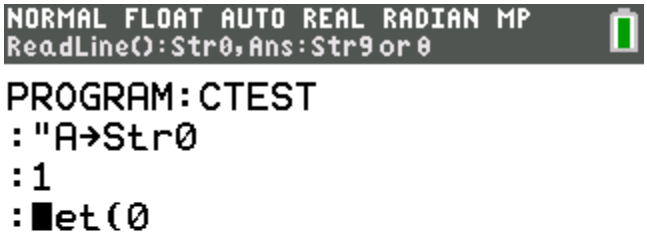

Fig. 3: Celtic's function preview feature.

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. If no input variables are necessary (though there are still output variables), it will say "NA" instead. Nothing will be listed of there are no input and 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. When storing a group, begin the string with the `*row(` For example:

```
:"FOO" -> Str0              //program FOO
:"rowSwap(FOO" -> Str0      //AppVar FOO
:"*row(FOO" ->              //group FOO
```

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

Warning: In almost all cases, Celtic will detect if you passed invalid arguments and return an error instead. However, in the (extremely) rare case that you tried to pass a string with the length of 0, it will not handle this. Don't worry, there shouldn't be any reason this would happen without you trying to make it happen.

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. This page also contains TI-OS errors and the corresponding values returned in the ErrorHandle command.

2.3.2 Celtic Errors

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.
..NULLVAR	The specified variable did not contain useful data.
..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.
..G:NT:FN	The specified group was not found.
..NT:A:GP	The file specified was not a group.
..SUPPORT	Whatever happened was not supported by Celtic.

2.3.3 TI-OS Errors

Value	Error
0	No Error
1	Overflow
2	DivBy0
3	SingularMat
4	Domain
5	Increment
6	Break
7	NonReal
8	Syntax
9	DataType
10	Argument
11	DimMismatch
12	Dimension
13	Undefined
14	Memory
15	Invalid
16	IllegalNest
17	Bound
18	GraphRange
19	Zoom
20	Label
21	Stat
22	Solver
23	Singularity
24	SignChange
25	Iterations
26	BadGuess
27	StatPlot
28	TolTooSmall
29	Reserved
30	Mode
31	LnkErr
32	LnkMemErr
33	LnkTransErr
34	LnkDupErr
35	LnkMemFull
36	Unknown
37	Scale
38	IdNotFound
39	NoMode
40	Validation
41	Length
42	Application
43	AppErr1
44	AppErr2
45	ExpiredApp
46	BadAddr
47	Archived
48	Version

continues on next page

Table 1 – continued from previous page

Value	Error
49	ArchFull
50	Variable
51	Duplicate

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"
```

`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"
```

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"
```

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.

2.6 Converting Sprites

2.6.1 Overview

Converting sprites by hand can be a pain, but thankfully it is possible to automate the process using a tool called convimg created by MateoConLechuga.

2.6.2 Setup

First you'll need to download the latest version of convimg from the GitHub releases page. You should choose the zip for your platform. Next you'll need to add convimg to your Path. To do this follow the instructions for your OS below:

Note: If you already have the CE C toolchain installed, chances are that convimg has already been installed and is in your Path. You can check this by opening a terminal and running *convimg –version*.

On Windows:

- Extract the zip to a file path without spaces.
- Open the start menu, then search and choose *Edit the system environment variables*.
- Click *Environment Variables* and select *Path* under *System variables*. Then press the *Edit* button.
- Click *New* and enter the path to convimg.
- Click *OK* and exit the editor. You may need to restart your apps (Terminals, IDEs) for the changes to take effect.

On Linux/macOS:

- Extract the zip to a file path without spaces.
- Open your system's rc file to add convimg to the path. This might be called `.bashrc`, `.zshrc`, or something else depending on your system.
- On macOS you might add something like this:

```
export PATH=$PATH:/path/to/convimg
```

- On Linux it might be something like this:

```
export PATH=/path/to/convimg:$PATH
```

- You may need to restart your apps (Terminals, IDEs) for the changes to take effect.

Once you've added convimg to your path, open a new terminal window and type `convimg --version` to ensure it been added correctly. If everything worked, you should see something like this:

```
tiny@tinyUbuntu:~$ convimg --version
convimg v9.0 by mateoconlechuga
```

2.6.3 Conversion

In order to convert your sprites, you should first create a folder with all the sprites for your project, to keep it organized. Having different sprites all over the place gets very confusing and is not a good habit to have. Once you have your sprites in a directory, you'll need to create a `convimg.yaml` file in that directory as well. This file has configuration options and information for convimg to convert your sprites. You can start out with a template file like this:

```
converts:
  - name: images
    palette: xlibc
    width-and-height: false
    images:
      - image.png

outputs:
  - type: basic
    include-file: images.txt
    converts:
      - images
```

Warning: If you are using an older version of convimg you may need to specify the type as "ice" instead of "basic". You can also pick up the latest non-release build of convimg by going to the GitHub Actions tab of the repo, selecting the most recent run, and downloading the version for your OS under "Artifacts".

If you want to convert file names which follow specific patterns, you can use glob patterns. For example, to convert all png files in a directory, you can do something like this:

```
images:
    - "*.png"
```

Note: It is necessary for the pattern to be in quotes, even though that is not necessary for image names to be in quotes.

Once you have completed your yaml, navigate to the directory with the sprites and the yaml in a terminal and run `convimg`. If all goes well you should have a text file as specified in your yaml (In the example it is `images.txt`) containing the converted sprites. It will look something like this:

```
image | 256 bytes
"FFFFFFFF000000000000000FFFFFFFFFFFFFFF0000... (Data continues)
```

You can then use the data in your programs by copying it into the source code or pasting it into a TI-BASIC IDE like SourceCoder.

For more documentation on convimg, check out the README here. For more info on glob patterns, look here.

CHAPTER

THREE

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, though it will still return the `..NUMSTNG` error, like the original Celtic 2 CSE. 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 occured; otherwise, contains an error code.

Errors:

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

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

Insets 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 occured; 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 occured; 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 occured; 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:

- `..INVAL: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:

- `..INVAL: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_low, fg_high, bg_low, bg_high)`

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_low`: low byte of foreground color.
- `fg_high`: high byte of foreground color.
- `bg_low`: low byte of background color.
- `bg_high`: 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, large_font, fg_low, fg_high, bg_low, bg_high, x, y); Str9 = text`

Displays colored text from `Str9` at `x` and `y` on the screen, using the OS large or small font.

Parameters:

- `large_font` = 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_low`: low byte of foreground color.
- `fg_high`: high byte of foreground color.
- `bg_low`: low byte of background color.
- `bg_high`: 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, large_font, 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.

`FillRect: 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 FillRect 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:

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

`PutSprite: det(21, x, y, width, height, string)`

This function draws a sprite at (x, y) with a width of width and a height of height, using data specified by string. For example, if the user specifies the string as 0, it will read the data from Str0. 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:

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

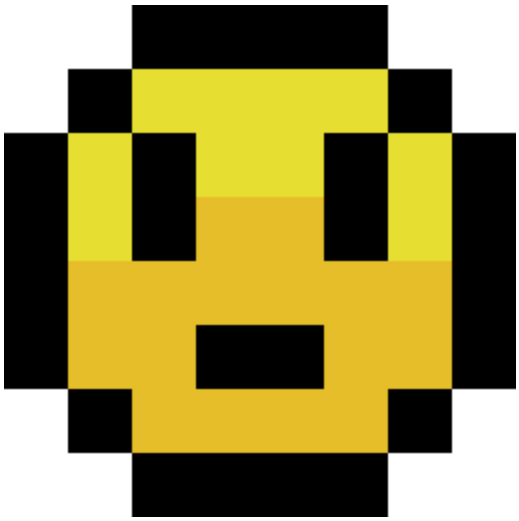

Fig. 1: A sample sprite.

```
[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:

```
"FFFF00000000FFFFFF00E6E6E6E600FF00E600E6E600E60000E600E5E500E600
00E5E5E5E5E5E50000E5E50000E5E500FF00E5E5E5E500FFFFFF00000000FFFF" -> Str9
```

More detailed instructions on converting sprites can be found here.

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.
- `string`: Which string variable to read the data from, 0-9.

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.

`GetStringWidth: det(54, font); Ans = string`

Gets the width of a string in pixels. This command works with both the OS large and small fonts.

Parameters:

- `font`: Whether to use the OS large or small font. 0 for small font, 1 for large font.
- `Ans`: Contains the string to be checked.

Returns:

- `Theta`: Contains the width of the string in pixels.

Errors:

- `..S:NT:ST` if `Ans` is not a string

`TransSprite: det(55, x, y, width, height, transparency, string)`

Draws a sprite where the user specifies the color to be interpreted as transparency. The rest of the functionality is identical to the PutSprite command.

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.
- `transparency`: Color in the sprite to be interpreted as transparency (0-255).
- `string`: Which string variable to read the data from, 0-9.

Returns:

- Draws a sprite with transparency 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.

`ScaleSprite: det(56, x, y, width, height, scale_x, scale_y, string)`

Draws a scaled sprite where the user can specify the scale value of both X and Y. The rest of the functionality is identical to the PutSprite command.

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`: Unscaled width of the sprite in pixels.
- `height`: Unscaled height of the sprite in pixels.
- `scale_x`: Width scaling value.
- `scale_y`: Height scaling value.
- `string`: Which string variable to read the data from, 0-9.

Returns:

- Draws a scaled sprite of the specified (scaled) 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.

`ScaleTSprite: det(57, x, y, width, height, scale_x, scale_y, transparency, string)`

Draws a scaled sprite where the user specifies the color to be interpreted as transparency. The rest of the functionality is identical to the PutSprite command.

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`: Unscaled width of the sprite in pixels.
- `height`: Unscaled height of the sprite in pixels.
- `scale_x`: Width scaling value.
- `scale_y`: Height scaling value.
- `transparency`: Color in the sprite to be interpreted as transparency (0-255).
- `string`: Which string variable to read the data from, 0-9.

Returns:

- Draws a scaled sprite with transparency of the specified (scaled) 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.

`ScrollScreen: det(58, direction, amount)`

Moves the screen a specified amount of pixels in a specified direction. The following directions and corresponding values are below:

Value	Direction
0	Up
1	Down
2	Left
3	Right

Parameters:

- `direction`: Direction to move the screen in, as seen in the table above.
- `amount`: The amount of pixels to move the screen.

Returns:

- Moves the screen in the specified direction.

`RGBto565: det(59, r, g, b)`

Converts an RGB color to a high and low byte that can be used in other Celtic functions.

Parameters:

- `r`: Red color value
- `g`: Green color value
- `b`: Blue color value

Returns:

- `Ans`: Low byte of the color.
- `Theta`: High byte of the color.

`DrawRect: det(60, low, high, x, y, width, height)`

Draws an unfilled rectangle of user-specified color, location, and size.

Parameters:

- `low`: Low byte of color
- `high`: High byte of color
- `x`: x location to begin drawing the rectangle, starting at the top-left corner of the screen
- `y`: y location to begin drawing the rectangle, starting at the top-left corner of the screen
- `width`: Width of the rectangle to draw
- `height`: Height of the rectangle to draw

Alternative method: `det(60, os_color, x, y, width, height)`

- `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 the unfilled rectangle.

`DrawCircle: det(61, low, high, x, y, radius)`

Draws an unfilled circle with the user specified color. The `x` and `y` arguments refer to the center point of the circle.

Parameters:

- `low`: Low byte of color
- `high`: High byte of color
- `x`: x location of the center of the circle, starting at the top-left corner of the screen
- `y`: y location of the center of the circle, starting at the top-left corner of the screen
- `radius`: Radius of the circle

Alternative method: `det(61, os_color, x, y, radius)`

- `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 the unfilled circle.

`FillCircle: det(62, low, high, x, y, radius)`

Draws a filled circle with the user specified color. The `x` and `y` arguments refer to the center point of the circle.

Parameters:

- `low`: Low byte of color
- `high`: High byte of color
- `x`: x location of the center of the circle, starting at the top-left corner of the screen
- `y`: y location of the center of the circle, starting at the top-left corner of the screen
- `radius`: Radius of the circle

Alternative method: `det(62, os_color, x, y, radius)`

- `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 the filled circle.

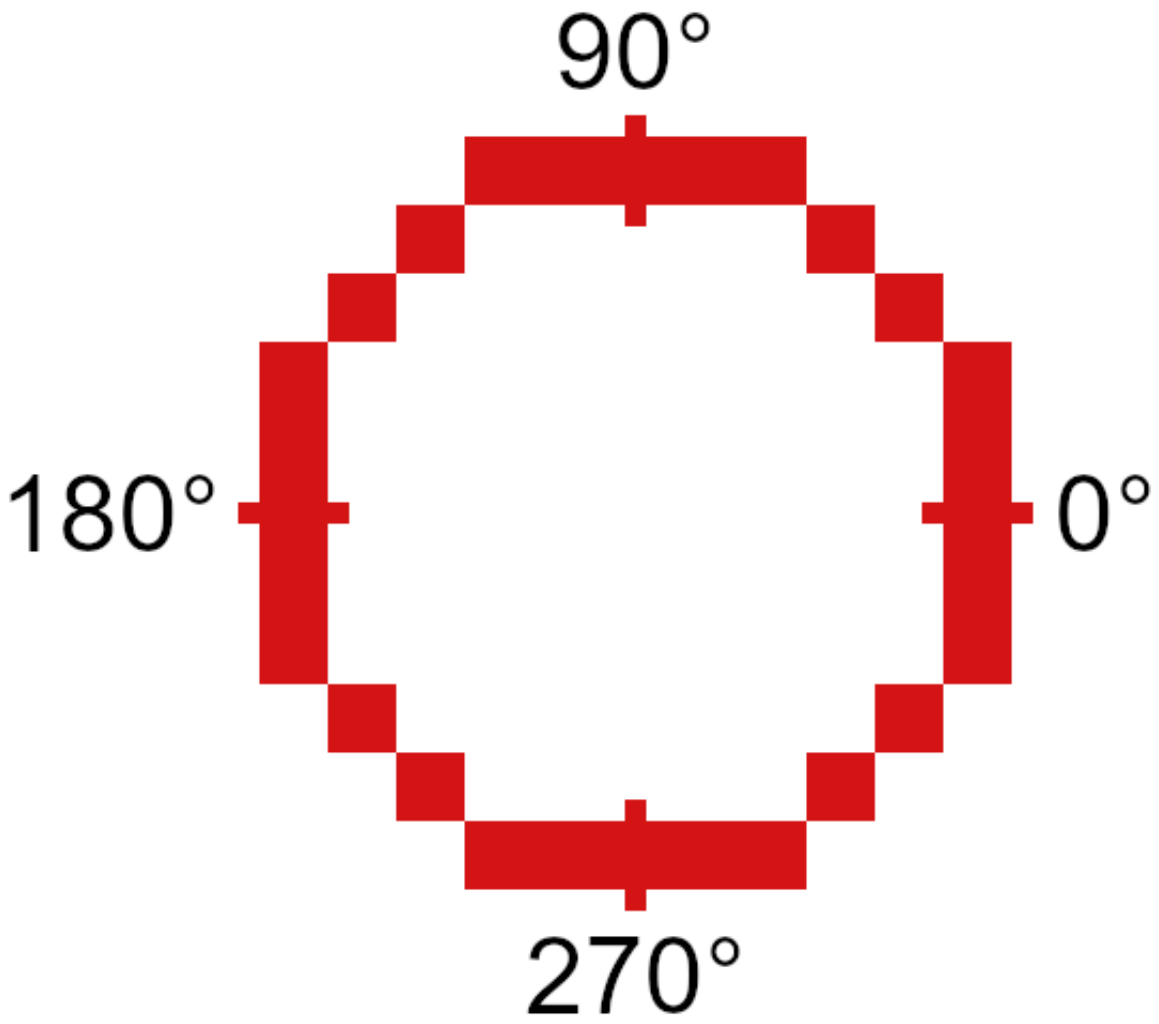

Fig. 2: A diagram showing various points along the circle.

`DrawArc: det(63, low, high, x, y, radius, start_angle, end_angle)`

Draws the outline of a circular arc. The `x` and `y` arguments refer to the center of the arc, which begins at `start_angle` and ends at `end_angle`. Angle values go from 0-360, and the end angle must be greater than the start angle. 0 degrees is the right most part of the circle, and goes counter-clockwise.

Parameters:

- `low`: Low byte of color
- `high`: High byte of color
- `x`: x location of the center of the arc, starting at the top-left corner of the screen
- `y`: y location of the center of the arc, starting at the top-left corner of the screen
- `radius`: Radius of the arc
- `start_angle`: Angle to begin drawing the arc at
- `end_angle`: Angle to finish drawing the arc at

Alternative method: `det(63, os_color, x, y, radius, start_angle, end_angle)`

- `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 the user-specified arc.

`DispTransText: det(64, font, low, high, x, y); Str9 = string to display`

Draws colored text with a transparent background in either the OS large or small font.

Parameters:

- `font`: Whether to use the OS large or small font. 0 for small font, 1 for large font.
- `low`: Low byte of color
- `high`: High byte of color
- `x`: x location to begin drawing the text at, starting at the top-left corner of the screen
- `y`: y location to begin drawing the text at, starting at the top-left corner of the screen
- `Str9`: String to display

Alternative method: `det(64, font, os_color, x, y)`

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

Colors:

- A list of colors can be found here.

ChkRect: det(65, x0, y0, width0, height0, x1, y1, width1, height1)

Checks if a rectangle intersects with another rectangle.

Parameters:

- `x0`: x coordinate of rectangle 0, starting at the top left corner of the screen
- `y0`: y coordinate of rectangle 0, starting at the top left corner of the screen
- `width0`: Width of rectangle 0
- `height0`: Height of rectangle 0
- `x1`: x coordinate of rectangle 1, starting at the top left corner of the screen
- `y1`: y coordinate of rectangle 1, starting at the top left corner of the screen
- `width1`: Width of rectangle 1
- `height1`: Height of rectangle 1

Returns:

- `Ans`: 0 if the rectangles do not intersect, and 1 if they do.

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 preceeded 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 preceeded by the `rowSwap(` token.
- `Str9`: The new name that you wish to rename the specified file to. It does not need to be preceeded 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_number); Str0 = variable to read`

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

Parameters:

- `string_number`: 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 preceeded by the `rowSwap(` token if you wish to read an appvar.

Returns:

- The contents of the specified variable in the string specified by `string_number`.

Errors:

- `..NT:EN:M` if there is not enough memory to create the string with the contents of the specified variable.
- `..NULLVAR` if the specified file contains no data.

`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.

SearchFile: det(52, offset); Str0 = file name, Str9 = search string

Search a file specified by `file name`, for a `search string`, beginning at the user-specified (0-indexed) `offset`.

Parameters:

- `offset`: Byte offset in the program to start searching, with 0 being the first byte of the program
- `Str0`: Name of the file to search in
- `Str9`: String to search for

Returns:

- `Theta`: The byte offset of the located string

Errors:

- `..E:NT:FN` if the string is not located
- `..INVAL:S` if the string is bigger than the program to search for

---

CheckGC: det(53); Str0 = variable name

Checks if the archiving of the file specified by `variable name` will trigger a Garbage Collect.

Note: If the file is already archived, the command will not say that archiving it will cause a Garbage Collect, regardless of size.

Parameters:

- `Str0`: Name of variable to check for

Returns:

- `Ans`: 0 if a Garbage Collect will not occur, and 1 if it will.

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 "$_{L}$FOO" (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 very 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.

UngroupFile: det(34, overwrite); Str0 = group name

This function ungroups the programs and AppVars from a group file specified in Str0. It will only apply to files that are programs and AppVars, not other types like lists. If overwrite is true (not 0), files which already exist will be overwritten. If it is false (0), files will be preserved. The group name (in Str0) must be preceded by the *row(token, as specified in General Syntax.

Parameters:

- `overwrite`: Whether or not to overwrite files that already exist when extracting.
- `Str0`: Contains the name of the specified group. The name must be preceded by the `*row(` token.

Returns:

- Ungroups all programs and AppVars from the specified group.

Errors:

- `..G:NT:FN` if the group specified does not exist.
- `..P:NT:FN` if no files in the group are able to be ungrouped (no programs or AppVars).

`GetGroup: det(35); Str0 = group name`

Puts the names of all program and AppVar files present in the specified group into `Str9`, separated by spaces. The names will be in the same order in the string as they are found in the group. The group name (in `Str0`) must be preceded by the `*row(` token, as specified in General Syntax.

Parameters:

- `Str0`: Contains the name of the specified group. The name must be preceded by the `*row(` token.

Returns:

- `Str9`: Contains a list of the names of all programs and AppVars in the group, separated by spaces.

Errors:

- `..G:NT:FN` if the group specified does not exist.
- `..P:NT:FN` if no files in the group are valid (no programs or AppVars).

`ExtGroup: det(36, item); Str0 = group name`

Extracts the program or AppVar specified by `item` from the group specified in `Str0`. If `item` is 1, it extracts the first program or AppVar, 2 extracts the second, and so on. This can be useful paired with `GetGroup` to figure out the order of the files in the group. If the file already exists with the same name, it will not be overwritten. The group name (in `Str0`) must be preceded by the `*row(` token, as specified in General Syntax.

> **Warning:** `item` only counts programs and AppVars, and ignores other types, like lists. If `item` is 2, it refers to the second **valid** file, not necessarily the second file including all types.

Parameters:

- `item`: The item in the group to extract. Only applies to programs and AppVars, and begins at 1.

Returns:

- Extracts the specified program or AppVar from the group.

Errors:

- `..G:NT:FN` if the group specified does not exist.
- `..E:NT:FN` if the specified item did not exist.
- `..P:IS:FN` if the program already exists.

GroupMem: det(37, item); Str0 = group name

Returns the size of the program or AppVar specfied by `item` from the group specified in `Str0`. `item` behaves the same way as in `ExtGroup`. The group name (in `Str0`) must be preceded by the `*row(` token, as specified in General Syntax.

Parameters:

- `item`: The item in the group to extract. Only applies to programs and AppVars, and begins at 1.

Returns:

- `theta`: The size of the specified program or AppVar from the group.

Errors:

- `..G:NT:FN` if the group specified does not exist.
- `..E:NT:FN` if the specified item did not exist.

BinRead: det(38, byte_start, number_of_bytes); Str0 = file name

Reads the contents of a file starting at `byte_start` for `number_of_bytes` bytes. `byte_start` is 0-indexed, meaning that the first byte of the program is 0, the second is 1, and so on. The output will be a hex string representing the bytes. For example, if the following bytes were in memory:

Byte	0	1	2	3	4
Data	EF	7B	66	6F	6F

`Str9` would contain this:

```
EF7B666F6F
```

Parameters:

- `byte_start`: The byte of the file to start reading from. It is 0-indexed, so the first byte of the file is 0, the second is 1, and so on.
- `number_of_bytes`: The number of bytes to read, starting at `byte_start`. You can also read past the end of the file.
- `Str0`: The name of the file to read from. For AppVars, the name should be preceded by the `rowSwap(` token.

Returns:

- `Str9`: Contains a text string of hex representing the bytes read.

BinWrite: det(39, byte_start); Str9 = hex string to write; Str0 = file name

Writes the hex bytes represented in `Str9` to the file specified by `Str0`, starting at `byte_start`. `byte_start` is 0-indexed, meaning that the first byte of the program is 0, the second is 1, and so on.

Parameters:

- `byte_start`: The byte of the file to start writing to. It is 0-indexed, so the first byte of the file is 0, the second is 1, and so on.

- `Str9`: Contains a text string of hex representing the bytes to write.
- `Str0`: The name of the file to write to. For AppVars, the name should be preceded by the `rowSwap(` token.

Returns:

- Writes the bytes specified in `Str9` to the specified file.

Errors:

- `..E:NT:FN` if `byte_start` is past the end of the file.
- `..INVAL:S` if there is not an even number of characters in the string or an invalid hex character is present.
- `..NT:EN:M` if there is not enough memory to complete the write.

`BinDelete: det(40, byte_start, number_of_bytes); Str0 = file name`

Deletes the `number_of_bytes` bytes from the file specified by `Str0`, starting at `byte_start`. `byte_start` is 0-indexed, meaning that the first byte of the program is 0, the second is 1, and so on.

Parameters:

- `byte_start`: The byte of the file to start deleting from. It is 0-indexed, so the first byte of the file is 0, the second is 1, and so on.
- `number_of_bytes`: The number of bytes to delete.
- `Str0`: The name of the file to delete from. For AppVars, the name should be preceded by the `rowSwap(` token.

Returns:

- Deletes the specified number of bytes from the specified file.

Errors:

- `..E:NT:FN` if `byte_start` is past the end of the file, or the number of bytes deleted would exceed the end of the file.

`HexToBin: det(41); Ans = hex string`

Outputs the binary equivalent of the input string in `Ans`. For example, if you did this:

```
:"464F4F424152
:det(41)
```

The result would be `"FOOBAR"`.

Parameters:

- `Ans`: Hex string to convert.

Returns:

- `Str9`: The converted string.

Errors:

- `..INVAL:S` if there is not an even number of characters in the string or an invalid hex character is present.

BinToHex: det(42); Ans = token string

Outputs the hex equivalent of the input string in Ans. For example, if you did this:

```
:"FOOBAR
:det(42)
```

The result would be "464F4F424152".

Parameters:

- Ans: Binary string to convert.

Returns:

- Str9: The converted string.

GraphCopy: det(43)

Copies the graph buffer to the screen.

Edit1Byte: det(44, string_number, target_byte, replace_byte)

Replaces the byte at target_byte of the specified string with the byte specified by replace_byte. target_byte is 0-indexed, meaning that the first byte of the program is 0, the second is 1, and so on. For example:

```
:det(44, 0, 0, 255)
```

This code will replace byte 0 of Str0 with the byte 255 in decimal, or 0xFF in hex.

Parameters:

- string_number: The string to modify. 0 is Str0, 1 is Str1, and so on.
- target_byte: The byte of the string to modify. It is 0-indexed, so the first byte of the file is 0, the second is 1, and so on.
- replace_byte: The byte to replace the target byte with, in decimal format. Can be 0 - 255 (0x00 - 0xFF).

Returns:

- Replaces the target byte of a string with the specified replacement byte.

Errors:

- ..E:NT:FN if the target byte does not exist in the string.

ErrorHandle: det(45); Ans = program to run

Executes BASIC code with an error handler installed. That means the code you execute can do anything it wants including divide by zero, and it will simply end the execution but an obvious system error will not trigger. Instead, this command will return with a value that indicates the error condition. This command has two different modes. If Ans contains a program name (beginning with the prgm token), it will run that program. If Ans contains program code, it will execute that code instead. This will also work with programs beginning with the Asm84CEPrgm token.

A list of return values and their corresponding errors can be found in the error codes section, under TI-OS Errors.

Note: When using ErrorHandle from the homescreen, it will not run BASIC programs, though it can still run programs beginning with the Asm84CEPrgm token.

Parameters:

- `Ans`: The name of the program to run, or TI-BASIC code to be executed

Returns:

- `Theta`: Contains the error code returned by the program, or 0 if no error occured.

Errors:

- `..NULLVAR` if the program is empty.
- `..SUPPORT` if the file is not a TI-BASIC program.

`StringRead: det(46, string, start, bytes);`

Works almost identically to BASIC's sub() command, except that the output will be in hexadecimal and two-byte tokens will read as two instead of one byte. It is particularly useful for extracting data from a string that may contain nonsensical data that simply needs to be manipulated. If you allow the start point to be zero, the size of the string in bytes is returned instead. For data manipulation, you should use the Edit1Byte command.

Parameters:

- `string`: Which string variable to read from, where 0 = Str0, 9 = Str9, and so on
- `start`: The byte of the string to begin reading at
- `bytes`: How many bytes to read

Returns:

- `Str9`: The extracted substring.
- `Theta`: The size of the string in bytes, if `start` was 0.

`HexToDec: det(47); Ans = hex`

Converts up to 4 hex digits back to decimal. If you pass a string longer than 4 digits, only the first four are read.

Parameters:

- `Ans`: Hex string to convert

Returns:

- `Theta`: Decimal integer converted from hex string.

Errors:

- `..INVAL:S` if an invalid hex digit is passed.

DecToHex: det(48, number, override)

Converts a number between 0 and 65535 to its hexadecimal equivalent. The number of hexadecimal output to the string will have its leading zeroes stripped so inputting 15 will result in "F" and 16 will result in "10". If override is 1, it will output all leading zeroes, which may be useful for routines that require four hex digits at all times but cannot spend the memory/time whipping up a BASIC solution to fill the missing zeroes.

Parameters:

- `number`: Decimal integer to convert
- `override`: 1 to output all leading zeroes, or 0 to not

Returns:

- `Str9`: Hex string converted from decimal integer.

EditWord: det(49, string, start, word)

This command, otherwise, works just like Edit1Byte. Its documentation is rewritten here for convenience. Replaces a word in some string variable, Str0 to Str9, with a replacement value 0 through 65535 starting at some specified byte (start is at 0). The string supplied is edited directly so there's no output. See Edit1Byte for more details.

The replacement is written in little-endian form and if the number is between 0 and 255, the second byte is written in as a zero.

Note: Note: A "word" in this sense is two bytes. Useful for editing a binary string which entries are all two bytes in length, such as a special string tilemap. You're required, however, to specify offset in bytes. Also know that all words are stored little-endian. That means that the least significant byte is stored before the most significant byte is.

Parameters:

- `string`: Which string variable to read from, where 0 = Str0, 9 = Str9, and so on
- `start`: The byte to start editing in the string
- `word`: The two bytes to rewrite

Returns:

- Modifies the string with the specified word

Errors:

- `..E:NT:FN` If the offset is past the end of the string

BitOperate: det(50, value1, value2, function)

Performs a bitwise operation between value1 and value2 using a supplied function value. It will only work with up to 16-bit numbers.

The different functions are below:

Value	Operation
0	NONE
1	AND
2	OR
3	XOR
4	Left Shift
5	Right Shift

If the numbers are out of bounds, then the function will exit out with an error. This command really helps mask out hex digits but if you use strings to store those digits, you'll need to use the HexToDec command for each value you need.

Parameters:

- `value1`: First value to perform bit operation with
- `value2`: Second value to perform bit operation with
- `function`: Which operation to perform, as seen in the table above

Returns:

- `Theta`: Result of the bit operation.

`GetProgList: det(51, type); Ans = search string`

This function will return a space-delimited string consisting of the names of programs, appvars, or groups that match partial name of the search string. Which is to say:

```
"TEMP
det(51, 0)
```

would return all program names that start with the characters "TEMP", which may be something like "TEMP001 " or "TEMP001 TEMP002 TEMP003 ", etc.

Value	File Type
0	Programs
1	AppVars
2	Groups

Note: This command is NOT to be confused with FindProg, which outputs a string consisting of files whose CONTENTS starts with the specified string. Also use the fact that the final name in the list is terminated with a space to make extracting names from the list easier. It also will not find hidden variables.

Parameters:

- `type`: The type of file to search for, as seen above
- `Ans`: String to find in file names

Returns:

- `Str9`: Filtered list of files

Errors:

- `..S:NT:FN` if `Ans` is not a string
- `..P:NT:FN` if no files were found containing the search string