Celtic CE Documentation

Release Beta 1.0

RoccoLox Programs, TIny_Hacker

Sep 20, 2022

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.

A pdf form of the documentation can be downloaded `here`.

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.

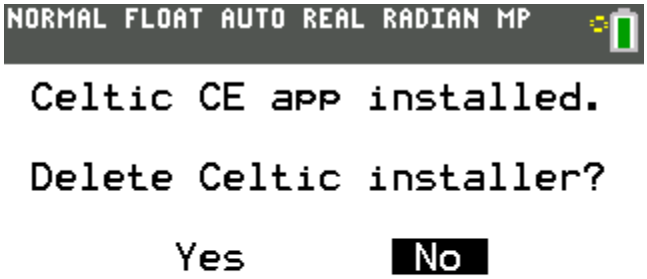

Fig. 1: The Celtic installer menu.

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 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.

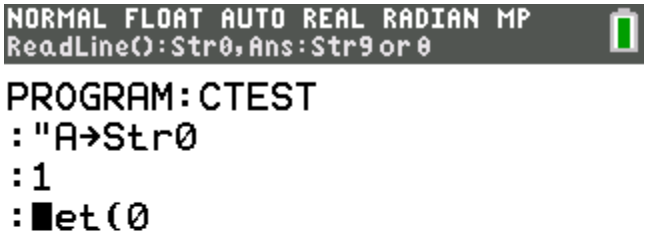

Fig. 3: Celtic's function preview feature.

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. 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 FOO
:"rowSwap(FOO" -> Str0      //AppVar FOO
```

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 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.
..SUPPORT	Whatever happened was not supported by Celtic.

2.4 Colors

2.4.1 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.

2.4.2 Colors

Refer to the DCS wiki for a list of pre-defined colors.

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. 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.

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.

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_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.

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.

To disable this mode, you should call `det(12, 300)` before exiting your program.

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.

> **Warning:** `Ans` must be under (not including) 255 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.