Doors CSE 8.1
Software Developers' Kit (SDK)


# DOORS CSE 8.1 SDK

Kerm Martian and Cemetech
http://dcs.cemetech.net

# TABLE OF CONTENTS

# CHAPTER 1
# INTRODUCTION

## Introduction to Developing for Doors CSE

Welcome to the Doors CSE SDK! This document is meant to be a comprehensive overview of the available tools and features for writing Doors CSE-based TI-BASIC and z80 assembly language programs. Since this document was written, it is likely that bug fixes, new features, and additional documentation has been created. Be sure to check http://dcs.cemetech.net for all of the information in this document and more, as well as the latest beta and final releases. You can discuss Doors CSE, developing for it, and any other questions and comments on the Cemetech forum at http://www.cemetech.net/forum.

This document discusses the available tools for BASIC programmers, then for ASM programmers. Doors CSE does not currently have features from Doors CS like the GUI system, the Associated Program (AP) system, and Shell Expansions, but future Doors CSE versions might offer such features. It concludes with a survey of places to get more information as well as the developer and end-user licenses.

## Using the Tools in This SDK

This SDK, besides being an informative document, is also a full set of tools to help you create BASIC programs and to build and assemble ASM programs for you.

Assembly programmers have a similar Header Creator, tailored to the needs of coders writing programmers, Appended Library Extensions (ALEs) or Shell Expansions (SEs), as well as a batch script to compile files, the TASM and Brass assemblers, and the BinPac8x Python-language linker for cross-platform compilation.

BASIC programmers should use TIFreak8x's Doors CSE 8 Icon Creator from http://www.cemetech.net/programs/index.php?mode=file&id=937.

### Structure of the SDK

The file structure of this SDK is as follows:

```
/SDK/
|       asm/                    Tools for z80 assembly programmers
+--     +--   exec/              Executable .8xp files end up here
|       +--   list/              Generated list files are put here
|       +--   source/            All source code should be here
|       +--   tasm/              Assemblers, linkers, and include files
|       +--   compile.bat        Windows compile script
|       +--   compile.sh         Linux/Mac compile script
```

```
|
+--  DCSE8_SDK.pdf          This document: SDK in PDF form
```

# CHAPTER 2
# OVERVIEW OF DOORS CSE BASIC TOOLS AND FEATURES

## Introduction to Doors CSE's BASIC Execution System

Doors CSE can execute BASIC programs from one of two major systems: from the Doors CSE desktop or from the desktop via the HomeRun feature that automatically catches any program execution and properly handles it. From a BASIC standpoint, these two execution methods are indistinguishable, and all Doors CSE features available to BASIC programs via one method will also be available via the other method. BASIC programs can take advantage of two major categories of Doors CSE features: metadata, such as icons, and BASIC library functions, including those from xLIBC and Celtic 2 CSE. This chapter will examine the available metadata and header formats in depth, while Chapter 3 will detail the available BASIC libraries.

## Basic Headers

BASIC headers allow the specification of metadata like icons and hide requests. All of the BASIC headers listed below are specifically designed to **not** interfere with normal program operation, so that programs with these headers can be run on any TI-84 Plus C Silver Edition calculator, even if Doors CSE is not loaded on the calculator. Note that if BASIC libraries are used (see Chapter 3), then this is no longer the case.

### "DCS" Header

The most commonly-used Doors CSE header is the standard "DCS" header. It allows a 16x16-pixel color icon to be specified. The correct form of these headers includes a double-quote character at the beginning of the hex string describing the icon. The hex string contains 256 (16x16) hex characters, with 0=transparent, 1=blue, 2=red, up to F=dark gray. The colors are the same as those used in the TI-OS/TI-BASIC color palette.

```
::DCS
:"256-char_hex_icon
:Program code
```

### Ignore Program Headers

Most program headers add additional data for shells to display. The Ignore Program headers both do the opposite: they hide a given program from Doors CSE, making it not display that program on the desktop. These headers are particularly useful for subprograms of main programs. Both possible Ignore Program headers are shown below; the upper one is for general programs, while the bottom one is for programs that use or process Ans.

```
:rand
:Program code
```

```
:Ans
:Program code
```

## BASIC Libraries

Doors CSE includes full supporting libraries for all of the most popular so-called Hybrid BASIC libraries, namely xLIBC and Celtic 2 CSE. Use of these libraries means that your program will not be able to be run on any calculator that does not have Doors CSE 8.0 or higher installed. On the other hand, the libraries afford your program much more power and control than pure BASIC programs can muster.

# CHAPTER 3
# BASIC LIBRARIES

## Overview

Doors CSE combines Celtic 2 CSE, based on work by Iambian Zenith, with xLIBC, a new library created by Patrick "tr1p1ea" Prendergast. Celtic 2 CSE primarily deals with creating, modifying, deleting, and using programs and AppVars; it also offers two sprite functions. xLIBC is primarily a graphics library.

Users can disable the Hybrid BASIC libraries from within Doors CSE's options screen, but if they are left enabled, and the HomeRun feature is enabled, then the BASIC libraries are available regardless of whether your BASIC program is run from within Doors CSE or from the TI-OS homescreen. This chapter will introduce each of the libraries, including the arguments and usage of each, as well as mention some notes and caveats for each library.

## xLIBC Functions

xLIBC is written by Patrick Prendergast, aka tr1p1ea. It is based on the concepts used to create xLIB for the TI-83 Plus/TI-84 Plus, included in Doors CS 7.2 for the monochrome calculators. xLIBC uses the TI-84 Plus C Silver Edition in half-resolution (160x240-pixel) mode, allowing one half of the LCD buffer to be modified off-screen while the other half is displayed. tr1p1ea has written an xLIBC Tutorial with an example program. Doors CSE has xLIBC compatibility based directly on tr1p1ea's original code. xLIBC can be detected with the following code sequence:

```
::DCS
:"256-char_hex_icon
:If 80>det([[20
:Then
:Disp "Get Doors CSE to run this:","http://dcs.cemetech.net
:Return:End
:Program code goes here
```

If this routine continues through to the program code, it guarantees that both xLIBC and Celtic 2 CSE are available for the program to use.

xLIBC uses a 256-color palette for all sprites, tilemaps, shapes, text, and fills. The palette is as follows:

| 0 0x00 | 1 0x01 | 2 0x02 | 3 0x03 | 4 0x04 | 5 0x05 | 6 0x06 | 7 0x07 | 8 0x08 | 9 0x09 | 10 0x0A | 11 0x0B | 12 0x0C | 13 0x0D | 14 0x0E | 15 0x0F | 16 0x10 | 17 0x11 | 18 0x12 | 19 0x13 | 20 0x14 | 21 0x15 | 22 0x16 | 23 0x17 | 24 0x18 | 25 0x19 | 26 0x1A | 27 0x1B | 28 0x1C | 29 0x1D | 30 0x1E | 31 0x1F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 0x20 | 33 0x21 | 34 0x22 | 35 0x23 | 36 0x24 | 37 0x25 | 38 0x26 | 39 0x27 | 40 0x28 | 41 0x29 | 42 0x2A | 43 0x2B | 44 0x2C | 45 0x2D | 46 0x2E | 47 0x2F | 48 0x30 | 49 0x31 | 50 0x32 | 51 0x33 | 52 0x34 | 53 0x35 | 54 0x36 | 55 0x37 | 56 0x38 | 57 0x39 | 58 0x3A | 59 0x3B | 60 0x3C | 61 0x3D | 62 0x3E | 63 0x3F |
| 64 0x40 | 65 0x41 | 66 0x42 | 67 0x43 | 68 0x44 | 69 0x45 | 70 0x46 | 71 0x47 | 72 0x48 | 73 0x49 | 74 0x4A | 75 0x4B | 76 0x4C | 77 0x4D | 78 0x4E | 79 0x4F | 80 0x50 | 81 0x51 | 82 0x52 | 83 0x53 | 84 0x54 | 85 0x55 | 86 0x56 | 87 0x57 | 88 0x58 | 89 0x59 | 90 0x5A | 91 0x5B | 92 0x5C | 93 0x5D | 94 0x5E | 95 0x5F |
| 96 0x60 | 97 0x61 | 98 0x62 | 99 0x63 | 100 0x64 | 101 0x65 | 102 0x66 | 103 0x67 | 104 0x68 | 105 0x69 | 106 0x6A | 107 0x6B | 108 0x6C | 109 0x6D | 110 0x6E | 111 0x6F | 112 0x70 | 113 0x71 | 114 0x72 | 115 0x73 | 116 0x74 | 117 0x75 | 118 0x76 | 119 0x77 | 120 0x78 | 121 0x79 | 122 0x7A | 123 0x7B | 124 0x7C | 125 0x7D | 126 0x7E | 127 0x7F |
| 128 0x80 | 129 0x81 | 130 0x82 | 131 0x83 | 132 0x84 | 133 0x85 | 134 0x86 | 135 0x87 | 136 0x88 | 137 0x89 | 138 0x8A | 139 0x8B | 140 0x8C | 141 0x8D | 142 0x8E | 143 0x8F | 144 0x90 | 145 0x91 | 146 0x92 | 147 0x93 | 148 0x94 | 149 0x95 | 150 0x96 | 151 0x97 | 152 0x98 | 153 0x99 | 154 0x9A | 155 0x9B | 156 0x9C | 157 0x9D | 158 0x9E | 159 0x9F |
| 160 0xA0 | 161 0xA1 | 162 0xA2 | 163 0xA3 | 164 0xA4 | 165 0xA5 | 166 0xA6 | 167 0xA7 | 168 0xA8 | 169 0xA9 | 170 0xAA | 171 0xAB | 172 0xAC | 173 0xAD | 174 0xAE | 175 0xAF | 176 0xB0 | 177 0xB1 | 178 0xB2 | 179 0xB3 | 180 0xB4 | 181 0xB5 | 182 0xB6 | 183 0xB7 | 184 0xB8 | 185 0xB9 | 186 0xBA | 187 0xBB | 188 0xBC | 189 0xBD | 190 0xBE | 191 0xBF |
| 192 0xC0 | 193 0xC1 | 194 0xC2 | 195 0xC3 | 196 0xC4 | 197 0xC5 | 198 0xC6 | 199 0xC7 | 200 0xC8 | 201 0xC9 | 202 0xCA | 203 0xCB | 204 0xCC | 205 0xCD | 206 0xCE | 207 0xCF | 208 0xD0 | 209 0xD1 | 210 0xD2 | 211 0xD3 | 212 0xD4 | 213 0xD5 | 214 0xD6 | 215 0xD7 | 216 0xD8 | 217 0xD9 | 218 0xDA | 219 0xDB | 220 0xDC | 221 0xDD | 222 0xDE | 223 0xDF |
| 224 0xE0 | 225 0xE1 | 226 0xE2 | 227 0xE3 | 228 0xE4 | 229 0xE5 | 230 0xE6 | 231 0xE7 | 232 0xE8 | 233 0xE9 | 234 0xEA | 235 0xEB | 236 0xEC | 237 0xED | 238 0xEE | 239 0xEF | 240 0xF0 | 241 0xF1 | 242 0xF2 | 243 0xF3 | 244 0xF4 | 245 0xF5 | 246 0xF6 | 247 0xF7 | 248 0xF8 | 249 0xF9 | 250 0xFA | 251 0xFB | 252 0xFC | 253 0xFD | 254 0xFE | 255 0xFF |

# Celtic 2 CSE

Celtic 2 CSE is intended to add many additional and powerful functions on top of the XLib routines. All Celtic functions start with the det() token. You can use the same sequence as above to make sure xLIBC and Celtic 2 CSE are present.

## Using XLib and Celtic Functions

Celtic 2 CSE catches certain commands as inputs for the application. The first argument for any of these caught commands is always the command's function number, which tells the application what it should do. These commands are the following:

```
:det(
:real(
```

The det( tokens will work as they normally do if used as they were originally intended. Doors CSE/Celtic 2 CSE is smart enough to know whether or not the command is for the TI-OS or for itself (with the sole exception of finding the determinant of a 1x1 matrix, but who would do that?). The real( token cannot be used as it was originally intended as per the precedent set by xLIBC, and Doors CSE uses the real( token to make it completely-compatible with xLIBC. The arguments after the function number are inputs for that particular function. If a real number (ie, not a complex number, and not a string) is specified, it can either be an immediate value (such as 42) or a letter variable containing the real number (such as K). Understand that this number MUST be real and not a complex number (such as 3+2i); such a value will make Doors CSE choke in a heartbeat. You can, however, pass negative numbers as arguments. The result will simply be the two's compliment of said number. Signed math is not performed.

If a string is specified in an argument, it can either be an immediate value (such as "this is a string") or a string variable (such as Str7). A few functions may make further restrictions, but they will be clearly explained in the command set listing. Sometimes one of these commands will NOT output a value. If the command set does not mention an output, do not assume there will be any.

Examples of commands are as follows (closing parentheses added for readability):

```
:det(3)                 //Grab the special characters
:real(2,0,0)            //Performs a getKey-like operation
```

The command reference later in this chapter define the output of each function. If some sort of number or string is output, it will be that function's result and can then be used as part of another function.

### Accessing Groups and AppVars

Some Celtic 2 CSE commands will require a string containing the name of a program file. Most commands will allow you to substitute the name of a program file for the name of an application variable (appvar). The main benefit of using appvars is that it doesn't clutter the programs menu, and they aren't readily accessible by the user. The downside is that applications that use appvars will EXPECT their own appvars to remain intact.

In order to tell a command that it should be accessing an appvar instead of a program file, insert the "rowSwap(" token at the beginning of the name. An example:

```
:"FOO"->Str9:1:det(1)           //Outputs the first line of program FOO
:"rowSwap(FOO"->Str9:1:det(1)   //Outputs the first line of AppVar FOO
```

**Notes**: You do not close the parenthesis on the rowSwap( token, as this will cause "File Not Found" errors at worst or if you're creating a file, an extra token at the end of the file that renders it inaccessible from any BASIC programs. Also, you can use lowercase letters to refer to AppVars but NOT programs. If you try to name a program using lowercase letters, you'll get similar results as closing that parenthesis, except much worse.

### What Not To Do

Obviously, don't try to use Celtic 2 CSE in ways it wasn't intended. You can abuse Doors CSE/Celtic 2 CSE to perform what you want, but neither I nor Iambian is responsible if Doors CSE crashes or freezes because you tried to input incorrect or illegal arguments.

### Error Catching and Handling

If a trappable error has occurred, Celtic 2 CSE will output a string containing the code of that error instead of letting the system run its course (either your standard ERR: message, or a system crash). The following codes describe the error:

| Long Code | Description |
|---|---|
| .P:IS:FN | A variable already exists and Celtic 2 CSE will not overwrite it |
| .NUMSTNG | Specified line is past end of file |
| .NULLSTR | An input of some kind exists but contains nothing. |
| .L:NT:FN | A line of code or an object was not found during a search. |
| .S:NT:FN | An input string was not found. |
| .S:FLASH | String is archived |
| .S:NT:ST | Variable is not a string |
| .NO:MEM | Not enough memory to complete operation |

| | |
|---|---|
| .E:2:LNG | Entry was too long. |
| .S:2:LNG | String was too long. |
| .NULLVAR | Some provided variable did not contain any useful content. |
| .P:NT:FN | A program or file you searched for doesn't exist. |
| .PGM:ARC | A program or file you selected is archived; cannot be edited. |
| .NULLINE | The line that was found didn't contain anything. |
| .T:NT:FN | Specified variable not found. |
| .P:IS:FN | Program exists, but shouldn't. |
| .SUPPORT | Whatever happened means it was not supported by Celtic 2 CSE. |

All that these "errors" indicate is that the normal, documented function of the command you used could not be run to completion. This may or may not be a good thing, especially if the commands are used in a function outside of its documented use, but may otherwise be perfectly safe. An example might be attempting to read a line out of a file simply to determine whether or not it even exists.

## BASIC Library Command Reference

### Celtic 2 CSE Routines

- ReadLine - det(0), Str0=program name, Ans=line number
- ReplaceLine - det(1), Str0=program name, Ans=line number, Str9=replacement
- InsertLine - det(2), Str0=program name, Ans=line number, Str9=contents
- SpecialChars - det(3)
- CreateVar - det(4), Str0=program/AppVar name
- ArcUnarcVar - det(5), Str0=program/AppVar name
- DeleteVar - det(6), Str0=program/AppVar name
- DeleteLine - det(7), Str0=program name, Ans=line number
- VarStatus - det(8), Str0=program name
- BufSprite - det(9,width,X,Y), Str9=sprite data
- BufSpriteSelect - det(10,width,X,Y,start,length), Str9=sprite data
- ExecArcPrgm - det(11,FN,NUMBER)
- DispColor - det(12,FG_LO,FG_HI,BG_LO,BG_HI)

### xLIBC Routines

- xLIBCSetup - real(0,FN,Value)
- UserVariables - real(1,FN,Uservar_Num[,Value])
- GetKey - real(2,FN[,Args])
- DrawMap - real(3,FN[,Args])
- DrawSprite - real(4,FN[,Args])
- ManagePic - real(5,FN[,Args])
- DrawString - real(6,FN[,Args])

- DrawShape - real(7,FN[,Args])
- xLIBCUtility - real(8,FN,Val)
- UpdateLCD - real(10)

## Celtic 2 CSE Library Routine Details

### ReadLine
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:ReadLine

**Description**
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. Useful for editors, curiosities, and verification. Otherwise, Ans=1 is the first line of the program, Ans=2 is the second, and so on.

**Technical Details**

**Arguments**
det(0)
**Str0**: Name of program to read from
**Ans**: Line number to read from (first line is "1")

**Outputs**
**Str9**: Contents of line read. Error "..NULLINE" if the line is empty. May also contain other error codes if conditions are wrong.

---

### ReplaceLine
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:ReplaceLine

**Description**
Replaces (overwrites) a line in a program or AppVar. Ans=1 is the first line of the program, Ans=2 is the second, and so on.

**Technical Details**

**Arguments**
det(1)
**Str0**: Name of program to read from
**Ans**: Line number to replace (first line is "1")
**Str9**: Contents to replace the line with

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

---

### InsertLine
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:InsertLine

**Description**
Inserts a line into a program or AppVar. Ans=1 is the first line of the program, Ans=2 is the second, and so on.

**Technical Details**

**Arguments**
det(2)
**Str0:** Name of program to write to
**Ans:** Line number to write to(first line is "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.

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

---

## SpecialChars
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:SpecialChars

**Description**
Generates a two-character string containing the STO character and double-quote character, in that order.

**Technical Details**

**Arguments**
det(3)
*(No other arguments)*

**Outputs**
**Str9**: Sto and doublequote characters, in that order. Use substrings to extract them. If using the standard version of Celtic, the string will be 9 characters long, the other 7 being junk. This should not affect the integrity of string just as long as you extract only the first two characters.

---

## CreateVar
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:CreateVar

**Description**
Create a program variable or an AppVar given a name.

**Technical Details**

**Arguments**
det(4)
**Str0:** Name of program or AppVar to create.

**Outputs**
**Str9**: Intact if nothing went wrong. Otherwise, an error code results. If successful, you will be able to read the first line of the newly created program as a null line.
**Str0**: Intact with program's name to be created

---

### ArcUnarcVar
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:ArcUnarcVar

**Description**
Archive/unarchive a program variable given a name.

**Technical Details**

**Arguments**
det(5)
**Str0**: Name of program or AppVar to move between Archive and RAM.

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

---

### DeleteVar
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DeleteVar

**Description**
Delete a program variable or an AppVar given a name.

**Technical Details**

**Arguments**
det(6)
**Str0**: Name of program or AppVar to delete.

**Outputs**
The indicated program or AppVar is deleted.

---

### DeleteLine
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DeleteLine

**Description**
Deletes a line from a program or AppVar. Ans=1 is the first line of the program, Ans=2 is the second, and so on.

**Technical Details**

**Arguments**
det(7)

**Str0**: Name of program to delete from
**Ans**: Line number to delete (first line is "1"). If Ans=0, this routine instead computes the number of lines in the program and returns them in Theta.

**Outputs**
Str9: Contains an error code if an error occurs.

---

### VarStatus
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:VarStatus

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

**Technical Details**

**Arguments**
det(8)
**Str0**: Name of program to examine

**Outputs**
**Str9:** Contains 9 byte output code.

1st character: "A"=Archived "R"=RAM "
2nd character: "V"=Visible "H"=Hidden
3rd character: "L"=Locked "W"=Writable "O"=AppVar
4th character: --RESERVED-- (filled with space char)

Five character string afterward is the size of data portion of variable.

Example: "AVL 00314" = Archived, visible, locked, and 314 bytes.

---

### BufSprite
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:BufSprite

**Description**
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.

**Technical Details**

**Arguments**
det(9,width,X,Y)
**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,Y** = Coordinates of top-left corner of sprite
**width** = Sprite width (height gets computed)

**Outputs**
**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.

---

### BufSpriteSelect
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:BufSpriteSelect

**Description**
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.

**Technical Details**

**Arguments**
det(10,width,X,Y,start,length)
**Str9** = Sprite data of one or more sprites as ASCII hex, one pixel per character. 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,Y** = Coordinates of top-left corner of sprite
**width** = Sprite width (height gets computed)
**start** = Offset into Str9 of the start of pixel data. *0 is the first character, not 1*
**length** = Length of the sprite data in characters

**Outputs**
Sprite drawn to LCD and stored to graph buffer.

---

### ExecArcPrgm
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:ExecArcPrgm

**Description**

Ans contains the name of the program you want to use in a string, then you use the function, then you run the generated temporary program file according to temp_prog_number. But, you'll need to know the function codes to further explain how this works:

0: Copy file to a file numbered by temp_prog_number
1: Delete a temporary file numbered by temp_prog_number
2: Delete all temporary files.

For example, say you wanted to copy an archived program "FOO" to prgmXTEMP002, do the following:

```
"FOO":det(11,0,2):prgmXTEMP002
```

If you wanted to do this to an ASM program "BAR" and have it copied to the 12th temporary file, do the following:

```
"BAR":det(11,0,12):Asm(prgmXTEMP012)
```

If you decided you are done with the copy of "FOO" from the first example and you wanted to delete it, do this:

```
det(11,1,2)
```

That will delete prgmXTEMP002 but will not touch the original file. If you want to clean up (get rid of all temp files), you can do the following:

```
det(11,2
```

Files will not be overwritten if you attempt to copy to a preexisting temp file.

**Technical Details**

**Arguments**

"PRGMNAME
det(11,function,temp_prog_number)

You can run the resultant temporary program via one of the following, depending on format:

`:prgmXTEMP0XX` or `:Asm(prgmXTEMP0XX)`

Note that only prgmXTEMP000 to prgmXTEMP015 are valid; anything above prgmXTEMP015 will return Undefined.

**Outputs**

See description.

## DispColor

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DispColor

### Description

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.

### Technical Details

### Arguments

det(12,FG_LO,FG_HI,BG_LO,BG_HI
**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

Because of TI-OS argument-parsing limitations, foreground and background colors must be provided as a sequence of two numbers in the range 0-255. Sample low and high bytes are below.

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

| | | |
|---|---|---|
| ALICEBLUE | 223 | 247 |
| ANTIQUEWHITE | 90 | 255 |
| AQUA | 255 | 7 |
| AQUAMARINE | 250 | 127 |
| AZURE | 255 | 247 |
| BEIGE | 187 | 247 |
| BISQUE | 56 | 255 |
| BLACK | 0 | 0 |
| BLANCHEDALMOND | 89 | 255 |
| BLUE | 31 | 0 |
| BLUEVIOLET | 92 | 137 |
| BROWN | 69 | 161 |
| BURLYWOOD | 208 | 221 |
| CADETBLUE | 244 | 92 |
| CHARTREUSE | 224 | 127 |
| CHOCOLATE | 67 | 211 |
| CORAL | 234 | 251 |

| | | |
|---|---|---|
| CORNFLOWERBLUE | 189 | 100 |
| CORNSILK | 219 | 255 |
| CRIMSON | 167 | 216 |
| CYAN | 255 | 7 |
| DARKBLUE | 17 | 0 |
| DARKCYAN | 81 | 4 |
| DARKGOLDENROD | 33 | 188 |
| DARKGRAY | 85 | 173 |
| DARKGREEN | 32 | 3 |
| DARKKHAKI | 173 | 189 |
| DARKMAGENTA | 17 | 136 |
| DARKOLIVEGREEN | 69 | 83 |
| DARKORANGE | 96 | 252 |
| DARKORCHID | 153 | 153 |
| DARKRED | 0 | 136 |
| DARKSALMON | 175 | 236 |
| DARKSEAGREEN | 241 | 141 |

| | | |
|---|---|---|
| DARKSLATEBLUE | 241 | 73 |
| DARKSLATEGRAY | 105 | 42 |
| DARKTURQUOISE | 122 | 6 |
| DARKVIOLET | 26 | 144 |
| DEEPPINK | 178 | 248 |
| DEEPSKYBLUE | 255 | 5 |
| DIMGRAY | 77 | 107 |
| DODGERBLUE | 159 | 28 |
| FIREBRICK | 4 | 177 |
| FLORALWHITE | 222 | 255 |
| FORESTGREEN | 68 | 36 |
| FUCHSIA | 31 | 248 |
| GAINSBORO | 251 | 222 |
| GHOSTWHITE | 223 | 255 |
| GOLD | 160 | 254 |
| GOLDENROD | 36 | 221 |
| GRAY | 16 | 132 |
| GREEN | 0 | 4 |
| GREENYELLOW | 229 | 175 |
| HONEYDEW | 254 | 247 |
| HOTPINK | 86 | 251 |
| INDIANRED | 235 | 202 |
| INDIGO | 16 | 72 |
| IVORY | 254 | 255 |
| KHAKI | 49 | 247 |
| LAVENDER | 63 | 231 |
| LAVENDERBLUSH | 158 | 255 |
| LAWNGREEN | 224 | 127 |
| LEMONCHIFFON | 217 | 255 |
| LIGHTBLUE | 220 | 174 |
| LIGHTCORAL | 16 | 244 |
| LIGHTCYAN | 255 | 231 |
| LIGHTGOLDENRODYELLOW | 218 | 255 |
| LIGHTGRAY | 154 | 214 |
| LIGHTGREEN | 114 | 151 |
| LIGHTPINK | 184 | 253 |

| | | |
|---|---|---|
| LIGHTSALMON | 15 | 253 |
| LIGHTSEAGREEN | 149 | 37 |
| LIGHTSKYBLUE | 127 | 134 |
| LIGHTSLATEGRAY | 83 | 116 |
| LIGHTSTEELBLUE | 59 | 182 |
| LIGHTYELLOW | 252 | 255 |
| LIME | 224 | 7 |
| LIMEGREEN | 102 | 54 |
| LINEN | 156 | 255 |
| MAGENTA | 31 | 248 |
| MAROON | 0 | 128 |
| MEDIUMAQUAMARINE | 117 | 102 |
| MEDIUMBLUE | 25 | 0 |
| MEDIUMORCHID | 186 | 186 |
| MEDIUMPURPLE | 155 | 147 |
| MEDIUMSEAGREEN | 142 | 61 |
| MEDIUMSLATEBLUE | 93 | 123 |
| MEDIUMSPRINGGREEN | 211 | 7 |
| MEDIUMTURQUOISE | 153 | 78 |
| MEDIUMVIOLETRED | 176 | 192 |
| MIDNIGHTBLUE | 206 | 24 |
| MINTCREAM | 255 | 247 |
| MISTYROSE | 60 | 255 |
| MOCCASIN | 54 | 255 |
| NAVAJOWHITE | 245 | 254 |
| NAVY | 16 | 0 |
| OLDLACE | 188 | 255 |
| OLIVE | 0 | 132 |
| OLIVEDRAB | 100 | 108 |
| ORANGE | 32 | 253 |
| ORANGERED | 32 | 250 |
| ORCHID | 154 | 219 |
| PALEGOLDENROD | 85 | 239 |
| PALEGREEN | 211 | 159 |
| PALETURQUOISE | 125 | 175 |
| PALEVIOLETRED | 146 | 219 |

| | | |
|---|---|---|
| PAPAYAWHIP | 122 | 255 |
| PEACHPUFF | 215 | 254 |
| PERU | 39 | 204 |
| PINK | 25 | 254 |
| PLUM | 27 | 221 |
| POWDERBLUE | 28 | 183 |
| PURPLE | 16 | 128 |
| RED | 0 | 248 |
| ROSYBROWN | 113 | 188 |
| ROYALBLUE | 92 | 67 |
| SADDLEBROWN | 34 | 138 |
| SALMON | 14 | 252 |
| SANDYBROWN | 44 | 245 |
| SEAGREEN | 74 | 44 |
| SEASHELL | 189 | 255 |
| SIENNA | 133 | 162 |
| SILVER | 24 | 198 |

| | | |
|---|---|---|
| SKYBLUE | 125 | 134 |
| SLATEBLUE | 217 | 106 |
| SLATEGRAY | 18 | 116 |
| SNOW | 223 | 255 |
| SPRINGGREEN | 239 | 7 |
| STEELBLUE | 22 | 68 |
| TAN | 177 | 213 |
| TEAL | 16 | 4 |
| THISTLE | 251 | 221 |
| TOMATO | 8 | 251 |
| TURQUOISE | 26 | 71 |
| VIOLET | 29 | 236 |
| WHEAT | 246 | 246 |
| WHITE | 255 | 255 |
| WHITESMOKE | 190 | 247 |
| YELLOW | 224 | 255 |
| YELLOWGREEN | 102 | 158 |

*Thanks to Shaun "Merthsoft" McFall for the original color.h data used for this table.*

**Outputs**

See description.

# xLIBC Functions

### xLIBCSetup

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:xLIBCSetup

**Description**

Utility functions used when setting up an xLIBC program. real(0,0) gets the current xLIBC version, real(0,1,<bool>) turns half-resolution mode on or off, and real(0,20,<bool>) turns fast (15MHz) mode on or off.

**Technical Details**

GetXLIBCVersion

**real(0,0):** Returns version in Ans

SetupGraphics

**real(0,1,1):** Enable half-resolution (160x240-pixel) mode
**real(0,1,0,REDRAW_STATUS):** Disable half-resolution (160x240-pixel) mode
**REDRAW_STATUS** = 1 to redraw the status area at the top of the screen, 0 otherwise.

This function will set the calculator to half h-resolution/interlaced mode. Please note that ALL xLIB drawing functions expect the calculator to be in this mode. THIS MUST BE TURNED OFF BEFORE EXITING YOUR PROGRAM OR THE TIOS WILL BE STUCK IN THIS MODE! Also remember to redraw the status bar up the top of the screen or the TIOS will look funny. To disable 160 mode and redraw the status bar:

```
real(0,1,0,1
```

(If you forget to set this mode you will see how xLIB buffers each side of GRAM. This might be useful for debugging.

SetSpeed

**real(0,2,0):** Disable fast (15MHz) mode, switching to 6MHz mode
**real(0,2,1):** Enable fast (15MHz) mode

Note that the default CPU speed is 15MHz

SetupColorMode

**real(0,3,VALUE):**
VALUE defines action to take:
0 = Full color
1 = 8-color
2 = ColorInvert
3 = ColorInvertOff (restore to normal)
4 = FillScreen

5 = SetColorOffset *(DCSE 8.1+ only)*

To invert the colors on the screen:

```
real(0,3,2
```

To restore the colors back to normal:

```
real(0,3,2
```

This is because inverting something twice will restore it back to normal. To restore the colors back to normal when you don't know the previous state of the screen:

```
real(0,3,3
```

To fill the screen (active GRAM side only) with a color from the standard xLIB 256-color palette:

```
real(0,3,4,COLOR,UPDATELCD
```

To set the COLOR_OFFSET value which is used by sprite and shape routines (note that VALUE is between 0-255)

This will change the color values per pixel of sprites as they are drawn to the LCD. It can be used for special effects (magic animations for example):

```
real(0,3,5,VALUE
```

---

**UserVariables**
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:UserVariables

**Description**
The TI-84+CSE version of xLIB (xLIBC) has some key differences from its previous counterparts. One of the most obvious ones is the facility to utilize 'internal user variables' for data storage and calculations as opposed to using TI-OS variables. Note that most functions have versions (usually version 'A' (like 'DRAWMAPA' for example) still take TI-OS variables as arguments. These user vars help xLIBC update data in the functions themselves and can save calculation time in your program.

IMPORTANT - User variables are only temporary and are stored at PlotsScreen (RAM: $987C). Please ensure that you are not using this memory area while executing your program as it may produce unexpected results.

**Technical Details**

GetUservar
**real(1,0,Uservar_Num):** Get the value of the given Uservar

---

SetUservar
**real(1,1,Uservar_Num,Value):** Set the value of the given Uservar to Value

---

AddToUservar
**real(1,2,Uservar_Num,Value):** Add Value to the value of the given Uservar

---

SubFromUservar
**real(1,3,Uservar_Num,Value):** Subtract Value from the value of the given Uservar

---

## GetKey

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:GetKey

**Description**

These routines have been significantly overhauled from the monochrome xLIB getKey function.

**Technical Details**

GetKey
**real(2,0,0):** Key code stored to Ans (as per _getCSC codes)

GetKeyCheckList
**real(2,0,GETKEY_CHECKNUM,GETKEY_KEYVAL,GETKEY_USERVAR,GETKEY_VALUE...etc)**
GETKEY_CHECKNUM = number of checks to perform (length of list to check against)
GETKEY_KEYVAL = key value to check for
GETKEY_USERVAR = Uservar to update if key value is pressed
GETKEY_VALUE = value to update Uservar by if key is pressed

Checks for a list of keypresses and updates the appropriate user variables

GetKeyArrows
**real(2,1,USERVAR_X,USERVAR_Y,VALUE_X,VALUE_Y):**
USERVAR_X = user variable index holding x coordinate to update (0-255)
USERVAR_Y = user variable index holding y coordinate to update (0-255)
VALUE_X = amount to update uservar by if left/right is pressed
VALUE_Y = amount to update uservar by if up/down is pressed

Checks for up,down,left,right arrows and updates the specified user variables:
If up is pressed then USERVAR_Y = USERVAR_Y - VALUE_Y
If down is pressed then USERVAR_Y = USERVAR_Y + VALUE_Y
If left is pressed then USERVAR_X = USERVAR_X - VALUE_X
If right is pressed then USERVAR_X = USERVAR_X + VALUE_X

GetKeyArrowsDiagonals
**real(2,2,USERVAR_X,USERVAR_Y,VALUE_X,VALUE_Y):**

Checks for up,down,left,right and diagonal arrows and updates the specified user variables:
If up is pressed then USERVAR_Y = USERVAR_Y - VALUE_Y
If down is pressed then USERVAR_Y = USERVAR_Y + VALUE_Y
If left is pressed then USERVAR_X = USERVAR_X - VALUE_X
If right is pressed then USERVAR_X = USERVAR_X + VALUE_X
If up+left is pressed then USERVAR_Y = USERVAR_Y - VALUE_Y, USERVAR_X = USERVAR_X - VALUE_X
If up+right is pressed then USERVAR_Y = USERVAR_Y - VALUE_Y, USERVAR_X = USERVAR_X + VALUE_X
If down+left is pressed then USERVAR_Y = USERVAR_Y + VALUE_Y, USERVAR_X = USERVAR_X - VALUE_X
If down+right is pressed then USERVAR_Y = USERVAR_Y + VALUE_Y, USERVAR_X = USERVAR_X + VALUE_X

### GetKeyArrowsCheckTile

**real(2,3,USERVAR_X,USERVAR_Y,VALUE_X,VALUE_Y,USERVAR_MAPWIDTH,COLLISIONTILE,MAPSTRING,X0,Y0,X1,Y1):**
USERVAR_MAPWIDTH = width of tilemap in tiles (uservar 0-255)
COLLISIONTILE = upper limit of walkable tiles (any tile less than this will be walkable)
MAPSTRING = string variable holding tilemap data (0-10)
X0 = left x coordinate of collision box
Y0 = top y coordinate of collision box
X1 = right x coordinate of collision box
Y1 = bottom y coordinate of collision box

This performs the same as the above GetKeyArrows function but will only update the user variables if the move is to a walkable tile in the supplied tilemap. The X0,Y0,X1,Y1 is used to specify a 'collision box' around the x/y coordinate. Anything inside this coordinate must be walkable for the function to update the user variables.

### GetKeyArrowsDiagonalsCheckTile

**real(2,4,USERVAR_X,USERVAR_Y,VALUE_X,VALUE_Y,USERVAR_MAPWIDTH,COLLISIONTILE,MAPSTRING):**

Refer to the GetKeyArrowsCheckTile function; this adds diagonal keypresses as well.

### GetKeyUDLRCheckTileList *(DCSE 8.1+ only)*

**real(2,5,USERVAR_X,USERVAR_Y,VALUE_X,VALUE_Y,USERVAR_MAPWIDTH,COLLISIONTILE,MAPSTRING,X0,Y0,X1,Y1**

USERVAR_MAPWIDTH = width of tilemap in tiles (uservar 0-255)
COLLISIONTILE = upper limit of walkable tiles (any tile less than this will be walkable)
MAPSTRING = string variable holding tilemap data (0-10)

X0 = left x coordinate of collision box
Y0 = top y coordinate of collision box
X1 = right x coordinate of collision box
Y1 = bottom y coordinate of collision box

This function is the same as the above however it will return information regarding any keypresses and any collided tiles in a 'real list' contained in with the format {KEY_PRESS, NUM_COLLIDED_TILES, COLLIDED_TILES_LIST} where:

KEY_PRESS = -1,0,1,2,3 = NOARROW,UP,DOWN,LEFT,RIGHT
NUM_COLLIDED_TILES = number of tiles collided against given the arguments in the call
COLLIDED_TILES_LIST = list of tiles collided against given the arguments in the call

GetKey8DirCheckTileList *(DCSE 8.1+ only)*
**real(2,6,USERVAR_X,USERVAR_Y,VALUE_X,VALUE_Y,USERVAR_MAPWIDTH,COLLISIONTILE,MAPSTRING,X0,Y0,X1,Y1**

Refer to the above function; adds diagonal keypresses as well.

---

## DrawMap

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DrawMap

**Description**
Draw a tilemap. Tile data is stored in strings as hex encoded ASCII. For example, a 4x4 tilemap array (base 10 numbers):

```
[01,01,01,01]
[01,95,95,01]
[01,44,44,01]
[01,01,01,01]
```

would translate to:

```
"01010101015F5F01012C2C0101010101"->Str0
```

NOTE: You must load tiledata into temp memory before drawing tilemaps. See ManagePic.

**Technical Details**

DrawMapA (TI-OS Values)
**real(3,0,X,Y,MAPWIDTH,MAPSTRING,XSTART,YSTART,XEND,YEND,UPDATELCD):**
X = map x in tiles
Y = map y in tiles
MAPWIDTH = width of tilemap
MAPSTRING = string variable holding tilemap data (0-10)
XSTART = start x tile position on LCD to draw from (0-19)

YXSTART = start y tile position on LCD to draw from (0-14)
XEND = end x tile position on LCD to draw from (0-19)
YEND = end y tile position on LCD to draw from (0-14)
UPDATELCD = 0/1 to update LCD after drawing

Example: To draw a tilemap starting at 0,0 in a 32x32 tilemap and drawn with a 1 tile border around the outside (18x12 tiles) with tiledata stored in TIOS string 9:

```
real(3,0,0,0,32,9,1,1,18,13,1
```

---

DrawMapB (Uservar Values)

**real(3,1,USERVAR_X,USERVAR_Y,USERVAR_MAPWIDTH,MAPSTRING,XSTART,YSTART,XEND,YEND,UPDATELCD):**

USERVAR_X = map x in tiles (uservar 0-255)
USERVAR_Y = map y in tiles (uservar 0-255)
USERVAR_MAPWIDTH = width of tilemap in tiles (uservar 0-255)
MAPSTRING = string variable holding tilemap data (0-10)
XSTART = start x tile position on LCD to draw from (0-19)
YXSTART = start y tile position on LCD to draw from (0-14)
XEND = end x tile position on LCD to draw from (0-19)
YEND = end y tile position on LCD to draw from (0-14)
UPDATELCD = 0/1 to update LCD after drawing

---

DrawMap_GetTileA (TI-OS Values)

**real(3,2,X,Y,MAPWIDTH,MAPSTRING,XOFFSET,YOFFSET):**

X = map x in tiles
Y = map y in tiles
MAPWIDTH = width of tilemap
MAPSTRING = string variable holding tilemap data (0-10)
XOFFSET = x offset for X value
YOFFSET = y offset for Y value

Returns tile at map(X,Y) in Ans

---

DrawMap_GetTileB (Uservar Values)

**real(3,3,USERVAR_X,USERVAR_Y,USERVAR_MAPWIDTH,MAPSTR,XOFFSET,YOFFSET):**

X = map x in tiles (uservar 0-255)
Y = map y in tiles (uservar 0-255)
MAPWIDTH = width of tilemap (uservar 0-255)
MAPSTR = string variable holding tilemap data (0-10)
XOFFSET = x offset for X value
YOFFSET = y offset for Y value

---

Returns tile at map(X,Y) in Ans

---

DrawMap_SetTile (TI-OS Values)
**real(3,4,X,Y,MAPWIDTH,MAPSTRING,TILEID,XOFFSET,YOFFSET):**
X = map x in tiles
Y = map y in tiles
MAPWIDTH = width of tilemap
MAPSTRING = string variable holding tilemap data (0-10)
TILEID = index of tile to write into map
XOFFSET = x offset for X value
YOFFSET = y offset for Y value

Sets tile at map(x,y) in tilemap string(0-10)

---

DrawMap_ReplaceTile (TI-OS Values)
**real(3,5,MAPSTRING,CHECKNUM,TILEID0,REPLACETILID0,TILEID1,REPLACETILID1...etc):**
MAPSTRING = string variable holding tilemap data (0-10)
CHECKNUM = number of checks to perform (length of list to replace)
TILEID0 = 1st tile id to search for
REPLACETILEID0 = tile to replace 1st tile id if found

Replaces all occurrences of tileidX with replacetileidX for CHECKNUM

---

DrawMap_GetSectionA (TI-OS Values)
**real(3,6,VALUE_XY,SECTIONSIZE):**
VALUE_XY = x/y value in *** pixels ***
SECTIONSIZE = section width/height in tiles

This function will give you the section in multiples of SECTIONSIZE depending on VALUE_XY stored to Ans. It effectively performs int((SECTIONSIZE * 8) / VALUE_XY) * SECTIONSIZE. This is useful for drawing different sections of a tilemap depending on x/y values

---

DrawMap_GetSectionB (USERVAR Values)
**real(3,6,VALUE_XY,SECTIONSIZE):**
VALUE_XY = x/y value in *pixels*
SECTIONSIZE = section width/height in tiles

This function is the same as above, however it references an internal uservar for the VALUE_XY argument

---

**DrawSprite**
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DrawSprite

**Description**
Renders a sprite to the screen.

**NOTE:** You must load tiledata into temp memory before drawing tilemaps. See ManagePic.

**Technical Details**

DrawSpriteA (TI-OS Values)
**real(4,0,X,Y,WIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,PICINDEXSTART,PICINDEX0,PICINDEX1...etc):**
X = x value
Y = y value
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, any color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
PICINDEXSTART = pic index to start drawing from (in following list)
PICINDEX0 = pic index in sprite data sheet

Note that sprites are drawn from 8x8 chunks (or tiles) and are stored in tile/sprite data appvars that must be loaded into memory before use (see MANAGEPIC). For sprites larger than 8x8 the function takes a list of arguments for each index of the sprite (in 8x8 chunks). The sprites are drawn *column* first, so a 16x16 sprite list layout:

```
---------
| 1 | 3 |
---------
| 2 | 4 |
---------
```

24x24 sprite list layout:

```
-------------
| 1 | 4 | 7 |
-------------
| 2 | 5 | 8 |
-------------
| 3 | 6 | 9 |
-------------
```

Example: To draw an 8x8 sprite at 32, 32 with a pic index of 155 that has a transparent color index of 248:

```
real(4,0,32,32,1,1,248,1,0,155
```

Example: To draw a 16x16 sprite at 32,32 with a pic index list of 12,13,14,15 that has a transparent color index 248:

```
real(4,0,32,32,2,2,248,1,0,12,13,14,15
```

You can use the PICINDEXSTART argument to specify which sprite to draw out of a list of sprites. This can be useful for drawing a differen sprite depending on a direction variable. For example drawing an 8x8 sprite where pic index 10=up, 11=down, 12=left & 13=right, and you have a direction variable "A" which holds 0=up, 1=down, 2=left & 3=right you can do:

```
real(4,0,32,32,1,1,248,1,A,10,11,12,13
```

This will select index 10 if A=0, 11 if A=1, 12 if A=2 & 13 if A=3 and so on.

### DrawSpriteB (Uservar Values)

**real(4,1,USERVAR_X,USERVAR_Y,WIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,PICINDEXSTART,PICINDEX0,PICINDEX1...etc):**
X = x value
Y = y value
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, any color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
PICINDEXSTART = pic index to start drawing from (in following list)
PICINDEX0 = pic index in sprite data sheet

See above, the only difference is that it will reference user variables for x/y.

### DrawSpriteList8x8A (TIOS Values)

**real(4,2,LISTCOUNT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,X0,Y0,PICINDEX0,X1,Y1,PICINDEX1...etc:**
LISTCOUNT = number of sprites in list
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, any color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
X0 = x value for 1st sprite in list

Y0 = y value for 1st sprite in list
PICINDEX0 = pic index for 1st sprite in list

This function draws a list of 8x8 sprites (max 32 sprites per call). Example: To draw 3 sprites at (10,10),(20,20),(30,30) with pic indicies 12,14,16 with a transparent index of 248:

```
real(4,2,3,0,0,248,1,10,10,12,20,20,14,30,30,16
```

### DrawSpriteList8x8B (Uservar Values)

**real(4,3,LISTCOUNT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,X0,Y0,PICINDEX0,X1,Y1,PICINDEX1...etc:**
LISTCOUNT = number of sprites in list
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, any color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
X0 = x value for 1st sprite in list
Y0 = y value for 1st sprite in list
PICINDEX0 = pic index for 1st sprite in list

This function is the same as above but it takes USERVAR values for X0,Y0,X1,Y1 etc

### DrawSpriteTileBGA (TIOS Values)

**real(4,4,X,Y,WIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,MAPWIDTH,MAPSTRING:**
X = x value
Y = y value
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, any color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
MAPWIDTH = width of tilemap
MAPSTRING = string variable holding tilemap data (0-10)

This function will draw the tiles for width*height at a specific sprite coordinate. The resultant tiles will be aligned to the map (it will only draw at intervals of 8-pixels). This is useful for restoring a tilemap background that has been overwritten by a sprite

### DrawSpriteTileBGB (Uservar Values)

**real(4,5,X,Y,WIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,MAPWIDTH,MAPSTRING:**
X = x value
Y = y value
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, any color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
MAPWIDTH = width of tilemap
MAPSTRING = string variable holding tilemap data (0-10)

This function is the same as above but references user variables instead of TI-OS values.

DrawSpriteCheckCollisionA (TI-OS Values) *(DCSE 8.1+ only)*
**real(4,6,COUNT,X,Y,W,H,CX0,CY0,CW0,CH0....CXn,CYn,CWn,CHn**
COUNT = number of coordinates to check against
X = master X to test list of coordinates against
Y = master Y to test list of coordinates against
W = master W to test list of coordinates against
H = master H to test list of coordinates against
CX0 = first X to test against master X
CY0 = first Y to test against master Y
CW0 = first W to test against master W
CH0 = first H to test against master H
CXn = nth X to test against master X
CYn = nth Y to test against master Y
CWn = nth W to test against master W
CHn = nth H to test against master H

nth should be equal to COUNT

This function will test the rectangular coordinates specified by the 'master X,Y,W,H' against each iteration of rectangular coordinates from CX0,CY0,CW0,CH0 to CXn,CYn,CWn,CHn and will return 0 or 1 in Ans where 0 = no collision between the 'master set' and the list and 1 = a collision with at least 1 set is found. A list of collided coordinate indexes in the user-defined 'real list' "XL" in the format:

{TOTAL_COORDS_COLLIDED,INDEX0...INDEXn} where:

TOTAL_COORDS_COLLIDED = total number of rectangular indexes in the call that collide with the 'master set'
INDEX0 = first index where 0 = the rectangular coordinates [CX0,CY0,CW0,CH0] and n would equal the [CX0,CY0,CW0,CH0]

Note that user-define list "XL" is overwritten if it already exists. For example, rectangles at:

X = 10
Y = 10
W = 8
H = 8

Tested against others where:

| | | |
|---|---|---|
| X = 15 | X = 48 | X = 8 |
| Y = 15 | Y = 32 | Y = 0 |
| W = 8 | W = 16 | W = 64 |
| H = 8 | H = 16 | H = 16 |

The call would be: `real(4,6,3,10,10,8,8,15,15,8,8,48,32,16,16,8,0,64,16`

And the result would be:

```
Ans = 1
XL = {1,2,0,2
```

Ans=1 = collision found
{1 = collision found (would be 0 if no collisions occurred)
2 = number of collisions found
0 = collision with index 0
2 = collision with index 2

Index 1 is the square for which there was no collision.

You can check if there is a collision by using:

```
If Ans
If LXL(1)
```

Or by a similar method.

### DrawSpriteSequentialListA (TI-OS Values) *(DCSE 8.1+ only)*

**real(4,8,X,Y,WIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,PICINDEXSTART,PICINDEX0**
X = x value
Y = y value
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)

HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent color index, and color in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
PICINDEXSTART = pic index to start drawing from (in following list)
PICINDEX0 = pic index in sprite data sheet

This function will draw a sprite that of any size as per the same fashion as DRAWSPRITEA with the only difference being that you DON'T need to specify each PICINDEX for a largesprite, rather you only need to specify the FIRST PICINDEX. This means that your 8x8 sprite chunks will need to follow each other in your TILEPIC in SEQUENTIAL ORDER. The makeup of a large sprite is the same:

16x16 sprite list layout:

```
---------
| 1 | 3 |
---------
| 2 | 4 |
---------
```

24x24 sprite list layout:

```
-------------
| 1 | 4 | 7 |
-------------
| 2 | 5 | 8 |
-------------
| 3 | 6 | 9 |
-------------
```

For both of the above you only need to specifiy the PICINDEX '1' (along with appropriate WIDTH/HEIGHT ETC) to draw. The advantage is that you save space in BASIC code, speed of execution and you can maximise the space in your TILEPICS. The drawback is that the layout of sprites requires more work when creating your TILEPICS. You can use the PICINDEXSTART argument to have largesprite 'frames' as each PICINDEX you supply will be the STARTING INDEX for each frame so having:

```
real(4,8,10,10,2,2,0,0,248,1,0,10,20
```

Will draw a 16x16 sprite with the 4 '8x8 chunks' starting at INDEX 10 (10,11,12,13). If you change the PICINDEXSTART argument to 1 then the 16x16 sprite will be made up of the 4 '8x8 chunks' starting at INDEX 20 (20,21,22,23).

DrawSpriteSequentialListB (Uservar Values) *(DCSE 8.1+ only)*

**real(4,9,USERVAR_X,USERVAR_Y,WIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,PICINDEXSTART,PICINDEX0**

X = x value (uservar 0-255)
Y = y value (uservar 0-255)
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent colour index, and colour in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
PICINDEXSTART = pic index to start drawing from (in following list)
PICINDEX0 = pic index in sprite data sheet

This function is the same as above however it takes references to internal USERVARS as opposed to values directly.

DrawSpriteTileBGListA (TI-OS Values) *(DCSE 8.1+ only)*

**real(4,10,LISTCOUNT,LISTWIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,MAPWIDTH,MAPSTRING,X0,Y0...Xn,Yn**

LISTCOUNT = number of tiles in list
WIDTH = width of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
HEIGHT = height of sprite in 8x8 chunks (an 8x8 sprite is 1, 16x16 is 2, 12x12 is also 2 etc)
XOFFSET = offset for x value
YOFFSET = offset for y value
TRANSINDEX = transparent colour index, and colour in the sprite that matches this will be drawn transparent (0-255)
UPDATELCD = 0/1 to update LCD after drawing
MAPWIDTH = width of tilemap
MAPSTRING = string variable holding tilemap data (0-10)
X0 = first X value to draw tile at
Y0 = first Y value to draw tile at
Xn = last X value to draw tile at
Yn = last Y value to draw tile at

This function will draw the tiles for width*height at the sprite coordinate listed from X0,Y0 to Xn,Yn. The resultant tiles will be aligned to the map (it will only draw at intevals of 8-pixels). This is useful for restoring a tilemap background that has been overwritten by a list of sprites. As mentioned this function is the same as DRAWSPRITETILEBG just with a list.

DrawSpriteTileBGListB (Uservar Values) *(DCSE 8.1+ only)*

**real(4,11,LISTCOUNT,LISTWIDTH,HEIGHT,XOFFSET,YOFFSET,TRANSINDEX,UPDATELCD,MAPWIDTH,MAPSTRING,X0,Y0...Xn,Yn**
MAPWIDTH = width of tilemap (uservar 0-255)
MAPSTRING = string variable holding tilemap data (0-10)
X0 = first X value to draw tile at (uservar 0-255)
Y0 = first Y value to draw tile at (uservar 0-255)
Xn = last X value to draw tile at (uservar 0-255)
Yn = last Y value to draw tile at (uservar 0-255)

This function is the same as above how it takes references to internal USERVARS as opposed to direct values in the call.

---

## ManagePic

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:ManagePic

**Description**
Necessary step before using DrawSprite or DrawMap.

**Technical Details**

LoadTilePic (TI-OS Values)
**real(5,0,PICSLOT):**
Ans = AppVar name (as string, no rowSwap( prefix)
PICSLOT = 0/1 pic slot (temp RAM) to load tilepic into (slot0=tiles 0-127, slot1=tiles128-255)

This function loads custom tilepic image appvars (name stored as a string in ANS) into temp RAM (slot0/1). These images are 128x64 pixels using the custom xLIBC palette. They each hold 128 * 8x8 tiles.

picslot0 = tiles loaded into 0-127
picslot1 = tiles loaded into 128-255

You must load a tilepic into temp RAM before using DRAWMAP or DRAWSPRITE. Example: To load a tilepic named "TESTILE" into slot0:

```
:"TESTTILE
:real(5,0,0
```

LoadBGPic (TI-OS Values)
**real(5,1,BGSLOT):**
Ans = AppVar name (as string, no rowSwap( prefix)
BGSLOT = 0/1 bg slot (temp RAM) to load bgpic into

This function loads custom bgpic image appvars (name stored as a string in ANS) into temp RAM (slot0/1). These images are 80x60 pixels using the custom xLIBC palette. These images are low resolution and can be useful for low detail backgrounds.

### DisplayBGPic (TI-OS Values)

**real(5,2,UPDATELCD):**
Ans = AppVar name (as string, no rowSwap( prefix)
UPDATELCD = 0/1 to update LCD after drawing

This function will display a custom bgpic image appvar (name stored as a string in ANS). To display a bgpic named "BGTEST":

```
:"BGTEST
:real(5,2,1
```

### DrawPicSectionA (TI-OS Values)

**real(5,3,X,Y,WIDTH,HEIGHT,BGSLOT,UPDATELCD):**
X = source X in bgpic
Y = source Y in bgpic
WIDTH = width of section to draw
HEIGHT = height of section to draw
BGSLOT = source BGPIC slot 0/1
UPDATELCD = 0/1 to update LCD after drawing

This function will draw a section of a BGPIC to a specified location on the screen. You must have loaded a BGPIC into temp RAM before using this function.

### DrawPicSectionB (Uservar Values)

**real(5,4,X,Y,WIDTH,HEIGHT,BGSLOT,UPDATELCD):**
X = source X in bgpic
Y = source Y in bgpic
WIDTH = width of section to draw
HEIGHT = height of section to draw
BGSLOT = source BGPIC slot 0/1
UPDATELCD = 0/1 to update LCD after drawing

This function is the same as above but takes user variables as arguments instead.

### LoadSingleTile (TI-OS Values)

**real(5,5,TILEID)**
Ans = Tile data
TILEID = id of tile to write data to (0-255)

This function will write data from a string stored in ANS to the specified TILEID in TEMPRAM. Tile data in ANS is stored as a 'HEX encoded string' (2 bytes per pixel) where each HEX pair represents a colour index in the standard xLIB palette. Tile data is arraged in columns and from left to right like so:

```
00 08 16 24 32 40 48 56
01 09 17 25 33 41 49 57
02 10 18 26 34 42 50 58
03 11 19 27 35 43 51 59
04 12 20 28 36 44 52 60
05 13 21 29 37 45 53 61
06 14 22 30 38 46 54 62
07 15 23 31 39 47 55 63
```

So the string in ANS would look like:

```
"000102030405060708091011121314151617181920212223242526272829303132333
435363738394041424344454647484950515253545556575859606162 63
```

This way you can manually write tile/sprite data to TEMPRAM without the need for TILEPIC's.

Note that tiles are 8x8 pixels in size.

DisplayBGPic32 (TI-OS Values) (DCSE 8.1+ only)
**"APPVARNAME**
**real(5,6,UPDATELCD**
UPDATELCD = 0/1 to update LCD after drawing

This function will display a custom 32 colour 160x120 image (scaled to fullscreen) appvar (name stored as a string in ANS). To display a bgpic named "BG32TEST":

```
"BG32TEST
real(5,6,1
```

---

## DrawString

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DrawString

**Description**
This function draws a colored string with a transparent background using the custom xLIBC palette, in the custom xLIBC 8x8 font. The NEWLINECHAR code can be used kind of like a carriage return (X is reset and Y=Y+8). The string data is stored in Ans.

**Technical Details**
**real(6,0,X,Y,COLOR,NEWLINECHAR,UPDATELCD)**:
Ans = Input string
X = x value

Y = y value
COLOR = color index (standard xLIBC palette)
NEWLINECHAR = ascii character code that indicates a newline. String moved down 1 line and x is reset
UPDATELCD = 0/1 to update LCD after drawing

---

## DrawShape

http://dcs.cemetech.net/index.php/DCSE:BasicLibs:DrawShape

**Description**

This family of functions draws and reads pixels and shapes. See the beginning of this chapter for the palette used for all functions in this family.

**Technical Details**

GetPixelA (16-bit, TI-OS Values)

**real(7,0,X,Y):**
X = x value
Y = y value

This function returns a 16-bit color code from LCD GRAM in Ans

GetPixelB (xLIBC Palette, TI-OS Values)

**real(7,1,X,Y):**
X = x value
Y = y value

This function returns a custom xLIBC palette color code from LCD GRAM in Ans

SetPixelA (16-bit, TI-OS Values)

**real(7,2,X,Y,COLORHIGH,COLORLOW,SIZE,UPDATELCD):**
X = x value
Y = y value
COLORHIGH = high 8-bits of 16-bit color code
COLORLOW = low 8-bits of 16-bit color code
SIZE = size of pixel to draw
UPDATELCD = 0/1 to update LCD after drawing

This function sets a 16-bit color pixel in LCD GRAM

SetPixelB (xLIBC palette, TI-OS Values)

**real(7,3,X,Y,COLOR,SIZE,UPDATELCD):**
X = x value
Y = y value
COLOR = color index (standard xLIBC palette)

SIZE = size of pixel to draw
UPDATELCD = 0/1 to update LCD after drawing

This function sets a customer xLIBC palette color pixel in LCD GRAM

InvertPixel (TI-OS Values)
**real(7,4,X,Y,SIZE,UPDATELCD):**
X = x value
Y = y value
SIZE = size of pixel to draw
UPDATELCD = 0/1 to update LCD after drawing

This function inverts a pixel in LCD GRAM

DrawLine (xLIBC palette, TI-OS VALUES)
**real(7,5,X0,Y0,X1,Y1,COLOR,UPDATELCD):**
X0 = left x value
Y0 = top y value
X1 = right x value
Y1 = bottom y value
COLOR = color index (standard xLIBC palette)
UPDATELCD = 0/1 to update LCD after drawing

This function draws a line from X0,Y0 to X1,Y1 using the custom xLIBC palette

InvertLine (TI-OS VALUES)
**real(7,6,X0,Y0,X1,Y1,UPDATELCD):**
X0 = left x value
Y0 = top y value
X1 = right x value
Y1 = bottom y value
UPDATELCD = 0/1 to update LCD after drawing

This function draws a line from X0,Y0 to X1,Y1 with inverted colors

DrawRectangle (xLIBC palette, TI-OS VALUES)
**real(7,7,X0,Y0,WIDTH,HEIGHT,COLOR,UPDATELCD):**
X0 = left x value
Y0 = top y value
WIDTH = width of rectangle
HEIGHT = height of rectangle
COLOR = color index (standard xLIBC palette)
UPDATELCD = 0/1 to update LCD after drawing

This function draws a rectangle starting at X0,Y0 for WIDTH,HEIGHT using the custom xLIBC palette

InvertRectangle (TI-OS VALUES)

**real(7,8,X0,Y0,WIDTH,HEIGHT,UPDATELCD):**

X0 = left x value
Y0 = top y value
WIDTH = width of rectangle
HEIGHT = height of rectangle
UPDATELCD = 0/1 to update LCD after drawing

This function draws a rectangle starting at X0,Y0 for WIDTH,HEIGHT with inverted colors

FillRectangle (xLIBC palette, TI-OS Values)

**real(7,9,X0,Y0,WIDTH,HEIGHT,COLOR,UPDATELCD):**

X0 = left x value
Y0 = top y value
WIDTH = width of rectangle
HEIGHT = height of rectangle
COLOR = color index (standard xLIBC palette)
UPDATELCD = 0/1 to update LCD after drawing

This function draws a filled rectangle starting at X0,Y0 for WIDTH,HEIGHT using the custom xLIBC palette

InvertFilledRectangle (TI-OS Values)

**real(7,10,X0,Y0,WIDTH,HEIGHT,UPDATELCD):**

X0 = left x value
Y0 = top y value
WIDTH = width of rectangle
HEIGHT = height of rectangle
UPDATELCD = 0/1 to update LCD after drawing

This function draws a filled rectangle starting at X0,Y0 for WIDTH,HEIGHT with inverted colors

DrawCircle (TIOS Values)

**real(7,11,XCENTRE,YCENTRE,RADIUS,COLOR,UPDATELCD:**

XCENTRE = centre x value
YCENTRE = centre y value
RADIUS = radius of circle
COLOR = color index (standard xLIBC palette)
UPDATELCD = 0/1 to update LCD after drawing

This function will draw a circle (outline only) with its centre at XCENTRE,YCENTRE for RADIUS using the custom xLIBC palette

DrawFilledCircle (TIOS Values)
**real(7,12,XCENTRE,YCENTRE,RADIUS,COLOR,UPDATELCD:**
XCENTRE = centre x value
YCENTRE = centre y value
RADIUS = radius of circle
COLOR = color index (standard xLIBC palette)
UPDATELCD = 0/1 to update LCD after drawing

This function will draw a filled circled with its centre at XCENTRE,YCENTRE for RADIUS using the custom xLIBC palette.

---

## xLIBCUtility
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:xLIBCUtility

**Description**
Assorted xLIBC utility functions.

**Technical Details**

GetLCDBuffer (TI-OS Values)
**real(8,0)**

This function returns 0/1 = which side of the GRAM buffer to mark as active for drawing in Ans

SetLCDBuffer (TI-OS Values)
**real(8,1,VALUE):**
VALUE = 0/1 which side if the GRAM buffer to mark as active for drawing

SetGRAMOffset (TI-OS Values)
**real(8,2,VALUE):'**
VALUE = value to offset the LCD by

This function will offset the LCD by VALUE. This is useful for screen shaking effects etc.

GetRand (TI-OS Values)
**real(8,3,VALUE):**
VALUE = (0-255) upper bound of random number

This function returns a random integer between 0-VALUE in Ans

---

## UpdateLCD
http://dcs.cemetech.net/index.php/DCSE:BasicLibs:UpdateLCD

**Description**
Updates the LCD.

**Technical Details**
**real(10)**

This function updates the LCD. Effectively it switches positions in LCD GRAM.

# CHAPTER 4
# OVERVIEW OF DOORS CSE ASM TOOLS AND FEATURES

Doors CSE offers extensive tools for burgeoning ASM programmers to explore the power and capability of the shell. This SDK contains everything you need to turn your source code into executable programs to load on your favorite calculator or emulator, as well as more than enough documentation to get you there. If you get stuck, remember to come to the Cemetech forum (http://www.cemetech.net) and ask any questions, explain any comments, or expound any criticisms that you may have.

This SDK is specifically tailored to work in 32-bit and 64-bit environments, under Windows 2000 through Windows 8 and above, on Mac OS X, and on most popular Linux distributions. There are some prerequisites, as noted in the appropriate sections below.

## Documentation: The Doors CS Wiki

While this extensive document contains a wealth of up-to-date, carefully-checked information, new features are always being added to Doors CSE, bugfixes are occasionally implemented, and mistakes are sometimes found. Be sure to frequent the Doors CS wiki at http://dcs.cemetech.net to view the latest documentation and look for new versions of Doors CS and Doors CSE.

## Assembling Suite and compile.bat/compile.sh

In the interest of making the process of creating programs for Doors CSE as painless and straightforward as possible, this SDK contains a complete toolchain to turn source code into .8xp files. Place your source code in the /asm/source folder; indeed, two sample programs have been included. If you have any include files, be sure to put them in the /asm/tasm folder. In a command prompt, you will be typing either "compile docde7" (on Windows) or "./compile.sh docde7" (on Linux) if your program was docde7.asm or docde7.z80. The script will compile your program with Brass. Executable .8xp files end up in the /asm/exec directory, and List files containing annotated source go in the /asm/list/ directory. To recap:

### Windows Users

To compile docde7.asm or docde7.z80 into DOCDE7.8xp, open a command prompt ([Win][R], cmd, [enter]), use **cd** to navigate to the \asm\ directory of this SDK, and then type:

```
compile docde7
```

(or `compile docde7.asm` or `compile docde7.z80`). You no longer need Python, unless you are using BinPac8x.

### Linux/Mac Users

To compile docde7.asm or docde7.z80 into DOCDE7.8xp, open a terminal, use **cd** to navigate to the /asm/ directory of this SDK, and then type:

./compile.sh docde7

(or `./compile.sh docde7.asm` or `./compile docde7.z80`). In some circumstances it may be necessary to chmod +x compile.sh. You **must** have the program "mono" to assemble using Brass; you no longer need Python unless you're using BinPac8x. Please consult your distribution's documentation about how to install packages.

## Doors CSE 8 Include File

The Doors CSE 8 include file (dcse8.inc) is optimized for use with the included compilation package. Simply #include "dcse8.inc" **after** you #include "ti83plus.inc", and everything should work properly. If there are any problems that you can't fix on your own, please report them on the Cemetech forum. The include files defines all of the routines in this document for ASM programmers.

## Doors CSE Tools for ASM Programmers

### Program Headers

The Doors CSE header for simple executables follows a standard form, new for Doors CSE 8.0 and differing widely from the format used in Doors CS 7.x. Without further ado, the most minimal Doors CS header:

```
; Program Name:
; Author:
; Version:
; Date:
; Written for Doors CSE 8.0 and higher (http://dcs.cemetech.net)

.nolist
#include "ti84pcse.inc"
#include "dcse.inc"
.list

        .org UserMem
BinaryStart:
        .db $EF,$11             ;OpenLib(                   2    2
        .db "D",$BB,$BF,$BB,$BF,$BB,$C2,$BB,$C3,"CSE",$11,$3F
                                        ;    14   16 (tokenized "DoorsCSE"
        .db $EF,$12,$3F         ;ExecLib                    3   19
        .db $D5,$3F             ;Return                     2   21
        .db tExtTok,tAsm84CPrgm,$3F ;                       3   24 total
HeaderStart:
        .dw ASMStart-HeaderStart        ;offset to code
        ; Header sections start here

        .dw 10
```

```
        .db ASMHEADER_FIELD_TYPE_LIB    ;== 3
        .db "DoorsCSE",8,0     ;Lib name, min major version, min minor version

        .dw SECTIONSIZE ;(excludes type byte)
        .db SECTIONTYPE
        .db ...data...

        .dw SECTIONSIZE ;(excludes type byte)
        .db SECTIONTYPE
        .db ...data...  ;0 or more header fields in total

        .dw 0     ;End of header field: 0 bytes of data
        .db $ff   ;End of header field: type 255
ASMStart:
.relocate UserMem
ProgramStart:
        ...code...
.endrelocate
.end
```

If you want to add features like icons and descriptions, you will need to add those sections to the header.

### Header Field Types

Note: Most of these fields are optional. The Type 3 header is not optional for Doors CSE 8.0+ programs.

**0: Description**. Data is variable-length: zero-terminated string. Short descriptions are recommended.

**1: Icon.** Data is variable length.

- Offset 0: Icon type (1 byte).
  - Type 0: 32*32-pixel icon, 2-bit color (4 colors), 8-byte (4-color) palette
  - Type 1: 32*32-pixel icon, 1-bit color (2 colors), 4-byte (2-color) palette
  - Type 2: 16*16-pixel icon, 4-bit color (16 colors), 32-byte (16-color) palette
- Offset 1: Palette (bytes as indicated above)
- Offset 1+(palette size): Icon width, in displayed pixels. 32 for icon types 0-2.
- Offset 2+(palette size): Icon height, in displayed pixels. 32 for icon types 0-2.
- Offset 3+(palette size): Icon data

**2: Author name.** Data is a zero-terminated, variable-length string. Short names are recommended.

**3: Required library.** Data is fixed-length at 10 bytes.

- Offset 0: Library name (8 bytes, zero-padded)
- Offset 8: Minimum required major library version (1 byte)

- Offset 9: Minimum required minor library version (1 byte)

**4: Half-Resolution.** If present, shell will set half-resolution mode before executing program. Zero-length data.

**255: End of Header.** Must be at the end of every header. Zero-length data.

### App Header

Doors CSE 8.1 and above can list and execute standard TI-OS Applications (Apps). While Doors CSE 8 can display unmodified Apps, you can also give your Apps special header fields that Doors CSE will recognize, and which will not interfere with the TI-OS running your App.

Doors CSE 8.1 recognizes a single header field, containing a 32x32-pixel icon. It is placed within the standard App header, and has field type 290 with size 2 bytes. The 2-byte data is a little-endian pointer to an icon formatted according to the icon guidelines shown above. The example below shows the 290 header field in action.

```
; App Name:
; Author:
; Version:
; Date:
; Written for Doors CSE 8.1 and higher (http://dcs.cemetech.net)

.binarymode intel                 ; TI-83+ Application
.nolist
#include "ti84pcse.inc"
.list

NUM_PAGES = 1
.defpage 0, 16*1024, $4000          ; Page 0 definition
; Add additional page definitions as needed

.page 0                             ; Start page 0
.echoln "-Page 0----------------------"
        ; Master Field
        .db     80h, 0Fh, 0, 0, 0, 0
        ; Signing Key ID
        .db     80h, 12h, 1, 15 ; 15 for the TI-84+CSE
        ;revision
        .db 80h,21h,8   ; 8
        .db 80h,31h,1   ; Pre-release
        ; Name
        .db     80h, 44h, "AoE2"
        ; Disable TI splash screen.
        .db     80h, 90h
        ; Pages
        .db     80h, 81h, NUM_PAGES
        ; Date stamp.  Apparently, the calculator doesn't mind if you put
        ; nothing in this.
        .db     03h, 22h, 09h, 00h
        ; Date stamp signature.  Since nothing ever checks this, there's no
        ; reason ever to update it.  Or even have data in it.
        .db     02h, 00
```

```
        ; Doors CSE 8 Icon
        .db 29h, 02h
        .dw AppIcon
        ; Final field
        .db     80h, 70h
ASMStart:
        ; Your App's code goes here
        bjump(_jforcecmdnochar)

AppIcon:
Icon_Start:
        .db 2
        ;Icon type: 32x32, 2-bit color
        .db $ff,$ff,$d6,$99,$8c,$51,$73,$6d
        .db $ad,$54,$31,$85,$52,$89,$20,$e3
        .db $62,$a7,$72,$44,$93,$27,$51,$83
        .db $c4,$09,$f5,$8e,$9a,$04,$9a,$c9          ;32-byte (16-color)
palette
        .db 32,32
        ;Image dimensions, for the sprite routine
        .db $00,$00,$01,$23,$34,$10,$00,$00
        .db $00,$01,$55,$33,$63,$56,$10,$00
        .db $00,$17,$24,$28,$97,$33,$71,$00
        .db $00,$7a,$89,$66,$95,$b6,$27,$00
        .db $06,$8c,$aa,$a9,$9b,$77,$32,$60
        .db $07,$da,$99,$aa,$8b,$77,$64,$70
        .db $47,$ba,$9a,$a9,$9b,$77,$87,$74
        .db $4b,$ec,$39,$9a,$cb,$57,$a7,$52
        .db $47,$ba,$2a,$cc,$ea,$95,$a8,$72
        .db $17,$7e,$3a,$dd,$ec,$b5,$85,$71
        .db $05,$be,$26,$ca,$9e,$55,$85,$70
        .db $04,$7b,$23,$be,$eb,$65,$55,$20
        .db $00,$55,$33,$3a,$f3,$55,$75,$00
        .db $00,$05,$63,$3f,$f6,$35,$50,$00
        .db $01,$26,$86,$8f,$f8,$58,$62,$10
        .db $00,$14,$23,$33,$33,$32,$41,$00
Icon_End:
        .fill 256, 0                            ;Your App must be >256 bytes,
or the TI-OS will reject it
                                                ;Remove this for Apps that are
already >256 bytes
.end
```

# CHAPTER 5
# ASM ROUTINE SUMMARY

Doors CS contains an extensive library of routines to be used by z80 Assembly programmers. It also provides many BASIC libraries, as summarized in Chapter 3 of this document. The available ASM routines fall into five major categories, all exhaustively detailed herein:

- Graphics Routines
  - ClearLCDFull
  - ColorLine
  - ColorPixel
  - ColorRectangle
  - DrawSprite_1Bit
  - DrawSprite_2Bit
  - DrawSprite_4Bit
  - DrawSprite_4Bit_Enlarge
  - DrawSprite_8Bit
- Math Routines
  - MultHE
  - MultDEBC
  - DivHLC
  - RandInt
- Utility Routines
  - RunProg

## Graphics Routines

### ClearLCDFull

http://dcs.cemetech.net/index.php/DCSE:ClearLCDFull

**Description**
Clears the entire LCD to white. Takes about 2.957M cycles.

**Technical Details**
(None)

**Inputs**
*None*

**Outputs**
Entire LCD is cleared to white.

**Destroyed**
All

---

### ColorLine

http://dcs.cemetech.net/index.php/DCSE:ColorLine

**Description**
Draws a color line routine to the LCD using Bresenham's line algorithm.

**Technical Details**
Beware of saving and restoring iy. If your program quits with a corrupted iy, Doors CSE and/or the TI-OS *will* crash.

**Inputs**
**de** = x0
**bc** = y0
**hl** = x1
**ix** = y1
**iy** = 16-bit color

**Outputs**
Line drawn to LCD.

**Destroyed**
All

---

### ColorPixel

http://dcs.cemetech.net/index.php/DCSE:ColorPixel

**Description**
Colors a pixel on the LCD at the given coordinates using the specified color.

**Technical Details**
Beware of saving and restoring iy. If your program quits with a corrupted iy, Doors CSE and/or the TI-OS *will* crash.

**Inputs**
**hl** = x
**de** = y
**iy** = 16-bit color

**Outputs**
Pixel drawn to LCD.

**Destroyed**
All

---

## ColorRectangle
http://dcs.cemetech.net/index.php/DCSE:ColorRectangle

**Description**
Colors or inverts a rectangular area of the LCD.

**Technical Details**
This routine generally colors in a rectangular area of the LCD. However, color bc=$1337 is a special case; this color makes the routine invert all the colors on the screen by XORing each 16-bit pixel with $FFFF. Also, this routine can only display rectangles at even coordinates (h=10 means top-left X=20, for example), and only even widths are possible (d=15 is 30 pixels wide). This limitation allows X coordinates and widths to fit in 8-bit registers.

**Inputs**
**h** = top-left x (note: X-coordinate divided by 2!)
**l** = top-left y
**d** = width
**e** = height
**bc** = 16-bit color. Color $1337 is a special case.

**Outputs**
Rectangular area colored or inverted on LCD.

**Destroyed**
All

## DrawSprite_1Bit

http://dcs.cemetech.net/index.php/DCSE:DrawSprite_1Bit

### Description

Draws a 1-bit (2-color) sprite to the LCD.

### Technical Details

This routine draws an arbitrary-sized sprite to the LCD. In Doors CSE 8.0, it is implemented to allow any 16-bit position and any 8-bit width and height for sprites. It will sanely handle sprites that are partially off-screen by not drawing them at all. In future Doors CSE versions, it will perform proper clipping to display the onscreen portions of partially off-screen sprites.

### Inputs

**de** = top-left x
**hl** = top-left y
**ix** = pointer to sprite data with the following format:

```
.dw pointer_to_palette
.db widthpx, heightpx
.db bitpacked_padded_rows...
```

The routine accepts any 16-bit x-coordinate and any 16-bit y-coordinate. Widths must be at most 255 pixels; heights must be at most 240 pixels. Note: After Doors CSE 8.0, this routine will properly handle transparency at the edges of sprites with widths not divisible by 8, and clipping of partially off-screen sprites.

### Outputs

Sprite drawn to the LCD.

### Destroyed

All

## DrawSprite_2Bit

http://dcs.cemetech.net/index.php/DCSE:DrawSprite_2Bit

### Description

Draws a 2-bit (4-color) sprite to the LCD.

### Technical Details

This routine draws an arbitrary-sized sprite to the LCD. In Doors CSE 8.0, it is implemented to allow any 16-bit position and any 8-bit width and height for sprites. It will sanely handle sprites that are partially off-screen by not drawing them at all. In future Doors CSE versions, it will perform proper clipping to display the onscreen portions of partially off-screen sprites.

**Inputs**
**de** = top-left x
**hl** = top-left y
**ix** = pointer to sprite data with the following format:

```
.dw pointer_to_palette
.db widthpx, heightpx
.db bitpacked_padded_rows...
```

The routine accepts any 16-bit x-coordinate and any 16-bit y-coordinate. Widths must be at most 255 pixels; heights must be at most 240 pixels. Note: After Doors CSE 8.0, this routine will properly handle transparency at the edges of sprites with widths not divisible by 8, and clipping of partially off-screen sprites.

**Outputs**
Sprite drawn to the LCD.

**Destroyed**
All

---

## DrawSprite_4Bit

http://dcs.cemetech.net/index.php/DCSE:DrawSprite_4Bit

**Description**
Draws a 4-bit (16-color) sprite to the LCD.

**Technical Details**
This routine draws an arbitrary-sized sprite to the LCD. In Doors CSE 8.0, it is implemented to allow any 16-bit position and any 8-bit width and height for sprites. It will sanely handle sprites that are partially off-screen by not drawing them at all. In future Doors CSE versions, it will perform proper clipping to display the onscreen portions of partially off-screen sprites.

**Inputs**
**de** = top-left x
**hl** = top-left y
**ix** = pointer to sprite data with the following format:

```
.dw pointer_to_palette
.db widthpx, heightpx
.db bitpacked_padded_rows...
```

The routine accepts any 16-bit x-coordinate and any 16-bit y-coordinate. Widths must be at most 255 pixels; heights must be at most 240 pixels. Note: After Doors CSE 8.0, this routine will properly handle transparency at the edges of sprites with widths not divisible by 8, and clipping of partially off-screen sprites.

**Outputs**
Sprite drawn to the LCD.

**Destroyed**
All

---

### DrawSprite_4Bit_Enlarge
http://dcs.cemetech.net/index.php/DCSE:DrawSprite_4Bit_Enlarge

**Description**
Draws a 4-bit (16-color) sprite to the LCD, and displays each pixel as a 2x2 pixel block.

**Technical Details**
This routine draws an arbitrary-sized sprite to the LCD. In Doors CSE 8.0, it is implemented to allow any 16-bit position and any 8-bit width and height for sprites. It will sanely handle sprites that are partially off-screen by not drawing them at all. In future Doors CSE versions, it will perform proper clipping to display the onscreen portions of partially off-screen sprites.

**Inputs**
**de** = top-left x
**hl** = top-left y
**ix** = pointer to sprite data with the following format:

```
.dw pointer_to_palette
.db widthpx, heightpx
.db bitpacked_padded_rows...
```

The routine accepts any 16-bit x-coordinate and any 16-bit y-coordinate. Widths must be at most 255 pixels; heights must be at most 240 pixels. Note: After Doors CSE 8.0, this routine will properly handle transparency at the edges of sprites with widths not divisible by 8, and clipping of partially off-screen sprites.

**Outputs**
Sprite drawn to the LCD.

**Destroyed**
All

---

### DrawSprite_8Bit
http://dcs.cemetech.net/index.php/DCSE:DrawSprite_8Bit

**Description**
Draws an 8-bit (256-color) sprite to the LCD.

**Technical Details**

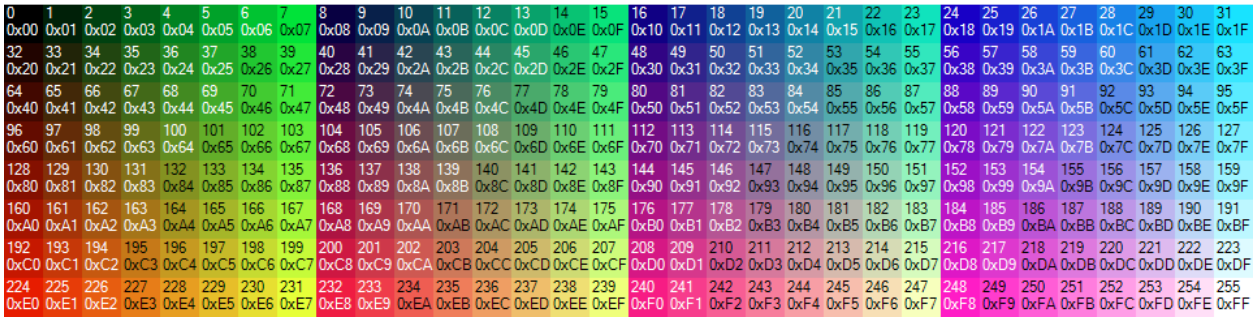
This routine draws an arbitrary-sized sprite to the LCD. In Doors CSE 8.0, it is implemented to allow any 16-bit position and any 8-bit width and height for sprites. It will sanely handle sprites that are partially off-screen by not drawing them at all. In future Doors CSE versions, it will perform proper clipping to display the onscreen portions of partially off-screen sprites.

**Inputs**

**de** = top-left x
**hl** = top-left y
**ix** = pointer to sprite data with the following format:

```
.dw pointer_to_palette
.db widthpx, heightpx
.db bitpacked_padded_rows...
```

If the pointer_to_palette is 0, then the routine will use the default palette where the high byte of the 16-bit color written is equal to the low byte. This creates the set of colors seen below. The routine accepts any 16-bit x-coordinate and any 16-bit y-coordinate. Widths must be at most 255 pixels; heights must be at most 240 pixels. Note: After Doors CSE 8.0, this routine will properly handle transparency at the edges of sprites with widths not divisible by 8, and clipping of partially off-screen sprites.




**Outputs**

Sprite drawn to the LCD.

**Destroyed**

All

## Math Routines

### MultHE

http://dcs.cemetech.net/index.php/DCSE:MultHE

**Description**
Multiplies two 8-bit numbers and returns a 16-bit product.

**Technical Details**
Special thanks to z80 Bits for the inspiration for this routine.

**Inputs**
**h** = multiplier
**e** = multiplicand

**Outputs**
**hl** = product

**Destroyed**
**b** = 0
c is left intact
a, hl, and de are destroyed

---

### MultDEBC

http://dcs.cemetech.net/index.php/DCSE:MultDEBC

**Description**
Multiplies two 16-bit numbers and returns a 32-bit product.

**Technical Details**
Special thanks to z80 Bits for the inspiration for this routine.

**Inputs**
**de** = multiplier
**bc** = multiplicand

**Outputs**
**dehl** = product

**Destroyed**
**a** = 0
bc, de, hl are destroyed

---

### DivHLC

http://dcs.cemetech.net/index.php/DCSE:DivHLC

**Description**
Divides a 16-bit dividend by an 8-bit divisor; returns a 16-bit quotient and an 8-bit remainder.

**Technical Details**
Special thanks to z80 Bits for the inspiration for this routine.

**Inputs**
**hl** = dividend
**c** = divisor

**Outputs**
**hl** = quotient
**a** = remainder

**Destroyed**
All

---

## RandInt

http://dcs.cemetech.net/index.php/DCSE:RandInt

**Description**
Generates a random integer between given bounds. Modifies (randData), aka (asm_prgm_size), a 2-byte value in RAM. For most programs, this is safe.

**Technical Details**
Modified from an older Ion-compatible routine in Doors CS 7.x.

**Inputs**
**b** = Upper bound (exclusive)

**Outputs**
**a** = Resulting number, 0 <= a < b

**Destroyed**
af, bc

## Utility Routines

### RunProg

http://dcs.cemetech.net/index.php/DCSE:RunProg

**Description**

Runs a program via Doors CSE's RunProg subsystem. This call can handle any program that Doors CSE itself can handle, including nostub BASIC and Hybrid BASIC, nostub and DCS ASM programs, plus any other filetype that gets added to Doors CSE after this document was written. Be very careful to take into account all the things that the running program could do to this program's SafeRAM, modes, and variables (see Outputs below).

**Technical Details**

**Inputs**

**hl =** Pointer to high (first) byte of VAT entry of given program or file.

**Outputs**

Runs the program. Be aware this may trash a variety of variables and settings, create new variables and settings, and switch many calculator modes, depending on the type of program run. Also note that Hybrid BASIC programs and an ASM program may completely destroy any SafeRAM in use, so be sure to either back up necessary variables

**Destroyed**

All (see Outputs)

# CHAPTER 6
# FURTHER READING

The most important place to get information about Doors CSE and Doors CS is the Doors CS Wiki, http://dcs.cemetech.net/. The second most important place is the Cemetech website and the Cemetech forums, where news about new versions, features, and programs is posted, and where users and programmers alike can ask questions about the shell. The front page is http://www.cemetech.net, and the Doors CS/Doors CSE subforum can be found at http://www.cemetech.net/forum/viewforum.php?f=9. General information about programming in z80 ASM and in TI-BASIC can be found at http://www.ticalc.org, and programmers are encouraged to ask programming questions and request advice on tutorials, project ideas, etc on the Cemetech forum.

As a last resort, you may hunt down my email address and drop me a line, but be advised that I will answer questions much faster when they're posted on the Cemetech forum, as I keep my eye on that much more frequently than I do my email.

Thanks for browsing this document, good luck with your TI programming endeavors, and I hope I get to hear from you on the Cemetech forum (http://www.cemetech.net/forum). Cheers!

# APPENDIX A
# LICENSE

Doors CSE is updated regularly to fix any reported bugs and compatibility issues, optimize size, and add new features. You can find all Doors CSE news at the Cemetech homepage, http://www.Cemetech.net. If you sign up as a Cemetech user, you can view the project page with beta editions and more at http://dcs.cemetech.net. You can download this and all future editions of Doors CSE from the Cemetech file archives or at the link above. If you have any comments, questions, complaints, or compliments on Doors CS, feel free to send me an email to dcse8@cemetech.net with the phrase "Doors CSE" in the subject line.



The full Doors CSE license is reproduced below:

## Doors CSE 8 End-User & Developer License

### A.1| Preamble

This license applies to any and all possible pieces of human- and machine-readable computer data, code, prose, graphics, and other materials in the assembly, basic, and other languages, including associated documentation, ideas, and intellectual property created, designed, and/or written by Christopher Mitchell, programming alias KERM MARTIAN. This document governs the use of the compiled data, code, source, graphics, and other materials and intellectual property of the official Doors CSE 8 alpha, beta, release candidate, and final releases. Any and all use and reuse of the Doors CSE code for any purpose including but not limited to an unofficial release of a compiled version by "Kerm Martian" must follow this agreement. Any attempt to use or reuse the source code or compiled code for release under "Kerm Martian" or another name must be explicitly approved by "Kerm Martian", except if such use or reuse has been previously approved by "Kerm Martian". Previous approval does not guarantee future approval, and "Kerm Martian" may choose to revoke any and all permissions granted to other coders regarding use of the Doors CSE code, data, or source including but not limited to circumstances of abuse or misuse.

By opening, downloading, or viewing this document, the executable binary program, the Software Developers' Kit, or the source code of DOORS CALCULATOR SHELL ENHANCED ("Doors CSE"), THE USER ("you") implicitly agree to the terms of this license agreement (LICENSE or AGREEMENT). If you do not accept the terms of this agreement, you are to

immediately delete this document and any related binary executable, documentation, information, and source code you have viewed, downloaded, or cached.

All legal rights accorded copyrighted or protected works not expressly covered in this document are reserved by "Kerm Martian".

### A.3| Scope

This license covers the electronically-encoded, hardcopy, and any other instance of the source and assembled code for the graphing calculator shell Doors CSE, its derivatives, and its modules. This license does NOT cover any program written to work with Doors CSE by either "Kerm Martian" or any other user, group, or organization. Certain portions of the code, including routines in whole or in part, have been used with the permission of the original third-party author(s). They may or may not be covered by original licenses. A user wishing to use that code should contact those authors for permission to use their code, or "Kerm Martian" may be able to contact the author on the user's behalf.

### A.4| Usage

Under abolutely no circumstances whatsoever may the source code of Doors CSE be recompiled in whole or in part and released by an individual, group, or third party other than "Kerm Martian" without express, expicit written permission from "Kerm Martian". Sections of code may be used in other published projects only with the written permission of "Kerm Martian". The source code of Doors CSE may be freely examined and reverse-engineered only for constructive purposes. It is explicitly illegal and contrary to this agreement to use any of the information covered directly or indirectly by this agreement for malicious or harmful purposes.

Optimizations, corrections, and bugfixes to this code may be submitted to the author, "Kerm Martian". Such items may be accepted or denied as additions or changes to the official source code maintained by "Kerm Martian" for official releases at "Kerm Martian"'s discretion. As a general rule, good, constructive suggestions will be almost definitely accepted.

Doors CSE itself is not for use for academic dishonesty or malicious or illegal activities. Such uses are a violation of this license agreement; such a violation nullifies the user's license to this program and requires its immediate removal as per section A.1 of this agreement.

### A.5| Liability

Cemetech and "Kerm Martian" hereby disclaim any and all responsibility for damage and/or injury to persons or property, both tangible and intangible, as a direct or indirect result of using Doors CSE 8. Among the implicit areas of non-liability deemed necessary for explicit statement here are "RAM Cleared" events, rendering of a mobile unit nonfunctional ("bricking"), and unwanted additional, removal, or modification of data on a device due directly or indirectly to Doors CSE 8, though all reasonable care has been taken to remove instability from this final release.

### A.6| Updates

No implied or express warranty is provided as to the frequency of updates to Doors CSE to add additional features, update existing features, repair bugs, or modify any other aspect or functionality of Doors CSE, its documentation, and its developer tools.