

Doors CS 7.1
Software Developers' Kit (SDK)



DOORS CS 7.1 SDK

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

TABLE OF CONTENTS

Table of Contents	2
Chapter 1 Introduction	9
Introduction to Developing for Doors CS	9
Using the Tools in This SDK	9
Structure of the SDK	9
Chapter 2 Overview of Doors CS BASIC Tools and Features	11
Introduction to Doors CS' BASIC Execution System	11
Basic Headers	11
"DCS" and "DCS6" Headers	11
Hybrid DCS-MOS Headers	12
Ignore Program Headers	12
BASIC Libraries	13
Chapter 3 BASIC Libraries	14
Overview	14
XLib Functions	14
Celtic III and PicArc Functions	15
Using XLib and Celtic Functions	15
Accessing Groups and AppVars	16
What Not To Do	17
Error Catching and Handling	17
Doors CS Functions	18
BASIC Library Command Reference	18
DCSB Libs Routines	18
Celtic Routines	19
PicArc Routines	20
XLib Routines	20
Omnicalc Routines	20
DCSB Lib Routine Details	21
StringWidth	21
ScanForPx	21
HomescreenToGraphscreen	22
EnumeratePicVars	22
ArcUnarcPic	23
DCSLibVersion	23

SimpleGUIMouse	24
PushGUIStack	25
PopGUIStacks	28
OpenGUIStack	28
CloseGUIStack	29
RenderGUI	29
GUIMouse	30
GUIMenu	31
PushAnsStack	33
PopAnsStack	34
ClearAnsStack	35
Celtic III Library Routine Details	36
ToggleSet	36
NumToString	36
GetListElem	37
GetArgType	37
ChkStats	38
LineRead	39
LineWrite	39
LineErase	39
LineReplace	40
FindProg	40
UngroupFile	41
GetGroup	41
ExtGroup	42
GroupMem	42
BinRead	42
BinWrite	43
BinDelete	43
HexToBin	43
BinToHex	44
FastCopy	44
ExecHex	44
Edit1Byte	45
ModeChange	45
IndexFile	46
LookupIndex	46
ErrorHandle	47
MatToStr	47
StringRead	48
HexToDec	48

DecToHex	48
EditWord	49
BitOperate	49
GetProgList	50
PicArc Library Routines	51
DBQuery1	51
TogglePic	52
GetPicGroup	52
ExtPicGroup	53
StringTile	53
PutSprite	54
ShiftScreen	54
DrawBox	55
FillMap	55
BoxShift	56
DrawText	57
CopyBox	58
XLib Compatibility Library Functions	59
ClearScreen	59
DrawSprite	59
DrawTileMap	60
RecallPic	61
ScrollScreen	62
ChngContrast	62
UpdateLCD	63
RunIndicator	63
GetKey	63
CreatePic	64
ExecArcPrgm	65
GetCalcVer	66
DrawShape	66
TextMode	67
CheckRAM	68
Omnicalc Compatibility Routines	69
Sprite	69
ExecAsm	69
Chapter 4 Overview of Doors CS ASM Tools and Features	70
Documentation: The Doors CS Wiki	70
Assembling Suite and Compile.bat/Compile.sh	70

Windows Users	70
Linux/Mac Users.....	71
Doors CS 7 Include File.....	71
Doors CS Tools for ASM Programmers.....	71
Types of Programs.....	71
Program Headers.....	71
The GUI API	73
The Associated Program System	74
Other Features	74
Chapter 5 The Doors CS GUI API	75
Working with the GUI API	75
Initializing the GUI.....	75
Adding GUI Elements.....	75
Space-Efficiency: Adding Multiple Elements at Once.....	76
Rendering the GUI.....	76
Accessing Modified Data	77
Removing GUI Elements	78
Closing the Stack	78
Example of a Simple GUI Program.....	78
GUI Memory Areas.....	79
GUI Stack Items	80
GUIRNull.....	81
GUIRLargeWin.....	81
GUIRSmallWin	82
GUIRFullScreenImg.....	82
GUIRText	83
GUIRWinButtons	83
GUIRWrappedText.....	83
GUIRButtonText	84
GUIRButtonImg	84
GUIRTextLineIn.....	84
GUIRRadio.....	85
GUIRCheckbox.....	85
GUIRByteInt.....	86
GUIRWordInt	86
GUIRHotspot.....	86
GUIRTextMultiline	87
GUIRSprite	87
GUIRLargeSprite.....	87

GUIRPassIn	88
GUIRScrollVert.....	88
GUIRScrollHoriz	89
GUIRBorder	89
GUIRRect	90
GUIRCUSTOM	90
GUIRMouseCursor	90
Chapter 6 Associated Program (AP) System.....	91
AP Routines	93
FileOpen.....	93
FileSave.....	93
FileSaveAs.....	94
Chapter 7 CALCnet ^{2.2}	95
Routines	95
Coding Practices	96
Memory Areas.....	96
Caveats	97
Finding Network Members.....	97
Sending	98
Receiving	99
More Information	99
Chapter 8 Shell Expansions.....	101
Why Write a Shell Expansion?	101
Technical Details.....	102
Persistent Storage	102
dModes: Return Values From SEs	102
Shell Expansion Header.....	103
Chapter 9 Routine Summary	104
Ion Compatibility Routines.....	105
iVersion.....	105
iRandom.....	105
iPutSprite	106
iLargeSprite.....	106
iFastCopy	106
iDetect	107
iDecompress.....	107
iGetPixel	108

Doors CS Legacy Routines	109
SmallWindow	109
SmallWindowErase	109
LargeWindow	109
LargeWindowErase	109
PlaySound	109
VDispHL	110
Pause	110
hDetect	111
DispLongInt	111
DPutMap	111
AP_GUIAVToTop	112
RunProg	112
Doors CS Graphical User Interface (GUI) Routines	114
OpenGUIStack	114
PushGUIStack	114
PushGUIStacks	115
GUIFindFirst	116
GUIFindNext	116
GUIFindThis	117
RenderGUI	118
GUIMouse	118
PopGUIStack	119
PopGUIStacks	120
CloseGUIStack	120
ResetAppPage	121
CALCnet ^{2,2} Routines	122
Cn2_Setup	122
Cn2_Setdown	122
Cn2_ClearRecBuf	122
Cn2_ClearSendBuf	123
Doors CS AP Routines	124
FileOpen	124
FileSave	124
FileSaveAs	125
MirageOS Routines	127
Chapter 10 Further Reading	129
Appendix A License	130
Doors CS 6 End-User License	130

A.1 Preamble.....	130
A.2 Availability	131
A.3 Scope	131
A.4 Usage.....	131
A.5 Liability.....	132
A.6 Updates	132

CHAPTER 1

INTRODUCTION

Introduction to Developing for Doors CS

Welcome to the Doors CS SDK! This document is meant to be a comprehensive overview of the available tools and features for writing Doors CS-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 CS, 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, then visits some of the specific subsystems available for coders, such as the Graphical User Interface or GUI system, the Associated Program (AP) system, and Shell Expansions. 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. BASIC programmers can use the HTML/Javascript Header Creator to design a header based on their specific needs, whether they need MirageOS cross-compatibility, what size icon, and whether they would like Doors CS to automatically unarchive and rearchive their subprograms.

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.

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

+--	basic/	Tools for TI-BASIC programmers
	+-- HeaderCreator.html	Header creator for BASIC programmers
+--	DCS7_SDK.pdf	This document: SDK in PDF

CHAPTER 2

OVERVIEW OF DOORS CS BASIC TOOLS AND FEATURES

Introduction to Doors CS' BASIC Execution System

Doors CS can execute BASIC programs from one of two major systems: from the Doors CS desktop, up until Doors CS 6.3 beta the only way to run a program, 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 CS 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 CS features: metadata, such as icons, descriptions, and lists of subprograms to automatically archive and unarchive, and BASIC library functions, including those from XLib, Celtic III, PicArc, parts of Omnicalc, and Doors CS' own powerful DCSB Libs. 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 descriptions, icons, subprograms, 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-83/TI-83+/TI-84+-series calculator, even if Doors CS is not loaded on the calculator. Note that if BASIC libraries are used (see Chapter 3), then this is no longer the case.

"DCS" and "DCS6" Headers

The most commonly-used Doors CS basic headers are the standard DCS and DCS6 headers. The former allows specification of an 8x8-pixel monochrome icon, with an option to also list subprograms to be automatically unarchived before this main program is run, and rearchived when it completes. The latter allows a 16x16-pixel monochrome icon with the same AutoUnarchive feature option. The correct form of these headers includes a double-quote character at the beginning of the hex string describing the icon. In Doors CS 5.0 and lower, this quote was not part of the header, so Doors CS will permanently AutoUpgrade any program omitting the double-quote to include it.

First, the DCS (8x8 icon) and DCS6 (16x16 icon) headers with no subprograms specified:

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

```
::DCS6  
:"64-char_hex_icon  
:Program code
```

Second, the same headers with the addition of the AutoUnarchive header. This requires the sequence “:A” directly after the header, then one or more lines indicating the 1 to 8 character name of a valid program, then a single line containing only a colon character: For example, both headers below describe a program with three subprograms, namely ZSUB1, ZSUB2, and SUBPRG5.

```
::DCS  
:"16-char_hex_icon  
::A  
:ZSUB1  
:ZSUB2  
:SUBPRG5  
:  
:Program code
```

```
::DCS6  
:"64-char_hex_icon  
::A  
:ZSUB1  
:ZSUB2  
:SUBPRG5  
:  
:Program code
```

Hybrid DCS-MOS Headers

For programmers that want MirageOS to still detect their BASIC programs, but don't want to lose the icon feature of Doors CS, the Hybrid headers are available. Both forms require the coder to specify a single 16x16-pixel icon; the first contains no description, while the second contains a description. Note that both lines of both headers start with an additional colon.

```
::  
::"64-CHAR 16x16 ICON  
:Program code
```

```
::"DESCRIPTION  
::"64-CHAR 16x16 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 CS, making it not display that program on the desktop. These headers are particularly useful for subprograms of main

programs, especially when coupled with the Archive section of the DCS or DCS6 header mentioned above in the main program to which the subprograms belong. 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 CS now includes full supporting libraries for all of the most popular so-called Hybrid BASIC libraries, namely XLib, Celtic III, PicArc, and part of Omnicalc. Use of these libraries means that your program will not be able to be run on any calculator that does not have at least Xlib, or Celtic, or Doors CS 7, depending on which library(ies) and routines you choose to use. On the other hand, though, the libraries afford your program much more power and control than pure BASIC programs can muster.

CHAPTER 3

BASIC LIBRARIES

Overview

Doors CS inherits most of its BASIC libraries from the popular Celtic III library package by Iambian Zenith; much of his code is used with his kind permission, plus additional bug fixes and optimizations that were added for Doors CS. Celtic III provides functions for its own libraries as well as emulating PicArc, a library dealing primarily with picture variables, XLib, an older and more widely-used Hybrid BASIC library, and a few functions from the classic Omnicalc. Doors CS adds its own set of functions called the DCSB Libs, which provide helpful functions not included in any of the other libraries in addition to powerful access to the DCS GUI system, previously available only to ASM programmers.

Users can disabled the Hybrid BASIC libraries from within Doors CS' options screen, but if they are left enabled, and the Parser Hook (also known as the HomeRun feature) is enabled, then the BASIC libraries are available regardless of whether your BASIC program is run from within Doors CS 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.

XLib Functions

Doors CS has XLib compatibility based on Celtic III, but with many bug fixes, stability improvements, and speed optimizations. In particular, the shape-drawing code has been completely overhauled, and many additional features and quirks have been added and corrected. XLib programs generally use the following sequence to detect whether XLib is installed, and if Doors CS programs use only XLib functions, they can use this method as well:

```
:DCS
:"16-char_hex_icon
:1:real(0
:If not(Ans:Then
:Disp "XLIB IS NOT",
"ONCALC. GET DCS7:",
"DCS.CEMETECH.NET"
:Return:End
:Program code goes here
```

If this routine continues through to the program code, it only guarantees that the XLib real()-based functions are available, not the Celtic III routines, PicArc functions, or DCSB Libs. Most XLib functions leave Ans unmodified, with the exception of a few that set a value into Ans.

Celtic III and PicArc Functions

Celtic III is intended to add many additional and powerful functions on top of the XLib routines, as well as supporting all of the original XLib functionality. All Celtic functions start with the `det()` token. As with XLib, Celtic III has its own detection sequence to check whether Celtic III is installed.

```
:DCS
:"16-char_hex_icon
:If det([[1:Then
:Disp "CELTIC III IS NOT",
"ONCALC. GET DCS7:",
"DCS.CEMETECH.NET"
:Return:End
:Program code goes here
```

If it passes through to the program code section, then you know that at least the `real()`, `identity()`, and `det()` commands are available, but there is no guarantee that the DCSB Libs are available. If you want to use the DCSB Libs as well, make sure that you instead use the DCSB Lib testing sequence in the next section.

Using XLib and Celtic Functions

Celtic III 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(
:identity(
:real(
```

The `det()` and the `identity()` tokens will work as they normally do if used as they were originally intended. Doors CS/Celtic III 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 xLib, and Doors CS uses the `real()` token to make it completely-compatible with xLib. 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 CS choke in a heartbeat. You can, however, pass negative numbers as arguments. The result will simply be the two's complement 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 (parentheses added for readability):

```
:det(0,"FOOBAR",4)      //uses TOGGLESET command to delete program FOOBAR
:identity(0,"FOO",0)    //Formats program FOO for image database usage.
:real(0,1)              //Clears LCD buffer, then updates the display.
```

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. For example:

```
det(29,det(31,det(28,"AAAA"),det(28,"5555"),2))
```

Explanation: The two inner det(28...) converts "AAAA" and "5555" from their hex notation into real numbers. They comprise the first two arguments of det(28...), a command used to perform a bitwise logic operation on its arguments. The 2 indicates that OR logic is to be applied; the result of that operation is 65535. The outer det(29...) takes one argument, and is used to convert a number to a hexadecimal string. The final result is the string "FFFF". That many nested det(...) commands requires careful checking, but for normal everyday use, it won't be confusing once you get down to writing your own code.

Accessing Groups and AppVars

Some 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 is as follows:

```
:det(5,"FOOBAR",1)      //Outputs the first line of program FOOBAR
:det(5,"rowSwap(FOOBAR",1) //Outputs the first line of appvar FOOBAR
```

Notes: You do not close the parenthesis on the rowSwap(token, as this can 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.

A select few commands require that the file specified is a group file. Unfortunately, you are forced to use the `"*row("` token as a prefix to tell Celtic III that it is a group, even though the command does not take any other type of file. You'll just have to deal with it.

```
: det(13,"*row(FOOBAR",1) //Outputs size of first extractable far from
                          //the group FOOBAR
```

The parenthesis note also applies to this.

What Not To Do

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

Error Catching and Handling

If a trappable error has occurred, Celtic III 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	Short	Description
.P:IS:FN	.1	A variable already exists and Celtic III will not overwrite it
.NULLSTR	.2	An input of some kind exists but contains nothing.
.L:NT:FN	.3	A line of code or an object was not found during a search.
.S:NT:FN	.4	An input string was not found.
.E:2:LNG	.5	Entry was too long.
.NULLVAR	.6	Some provided variable did not contain any useful content.
.P:NT:FN	.7	A program or file you searched for doesn't exist.
.PGM:ARC	.8	A program or file you selected is archived; cannot be edited.
.NULLINE	.9	The line that was found didn't contain anything.
.STR:ARC	.A	A string variable that was in use was archived.
.G:NT:FN	.B	A group that you was looking for wasn't found.
.NT:EN:M	.C	Not enough memory to complete operation
.NT:A:ST	.D	Input provided was not a string variable
.NT:REAL	.E	Input provided was not a real number
.NT:A:GP	.F	Input was not group file. Forgot the prefix?
.2:M:ARG	.G	Too many arguments were passed to Celtic III. Check your args.
.I:NT:FN	.H	Pic file not found.
.D:INVAL	.I	Database is not compatible with this version.
.E:NT:FN	.J	Database entry number was not found (too great?)
.INVAL:S	.K	String used was invalid for some reason.
.INVAL:A	.L	Arguments used was invalid. Check number of arguments used
.NT:A:LS	.M	The argument supplied was not a list
.SUPPORT	NS	Whatever happened means it was not supported by Celtic III.

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.

Doors CS Functions

Doors CS expands upon all of the other BASIC libraries by adding additional functions, such as commands to manipulate stacks of Ans variables, perform collision detection, work with text and strings, and create quick, attractive menus. Most importantly, it also provides near-full access to the Doors CS GUI system, a powerful tool available only to assembly programmers in previous versions of Doors CS. Like xLib and Celtic, Doors CS libraries have their own unique detection sequence:

```
: :DCS
: "16-char_hex_icon
: If 1337=det([[42]:Then
: Disp "DOORS CS 7 NOT",
: "ONCALC. GET DCS7:",
: "DCS.CEMETECH.NET"
: Return:End
: Program code goes here
```

Note that Doors CS uses a similar sequence to Celtic. In fact, `det([[42]])` will return 0 if Celtic III only is installed, 42 if neither Celtic III is installed, and 1337 if Doors CS is installed and providing DCSLib, Celtic, PicArc, Omnicalc, and XLib support.

Please carefully read the routine documentation for the non-GUI-related functions. For information about the GUI stack functions you should refer to the [GUI API chapter](#) of this SDK, which explains opening, pushing, rendering, using the mouse, popping, and closing.

BASIC Library Command Reference

DCSB Libs Routines

- [StringWidth](#) - sum(0,"String of token(s)")
- [ScanForPx](#) - sum(1,X,Y,direction) dir: 1=up, 2=down, 3=left, 4=right
- [HomescreenToGraphscreen](#) - sum(2,ULCD) ULCD: 0=no LCD update, 1=copy to LCD
- [EnumeratePicVars](#) - sum(3)
- [ArcUnarcPic](#) - sum(4,Pic#,desired) desired: 0=in RAM, 1=in Archive
- [DCSLibVersion](#) - sum(5)
- [SimpleGUIMouse](#) - sum(6,startX,startY)
- [PushGUIStack](#) - sum(7,type,{args...})
- [PopGUIStacks](#) - sum(8,numItems)
- [OpenGUIStack](#) - sum(9)
- [CloseGUIStack](#) - sum(10)

Doors CS 7 SDK

- [RenderGUI](#) - sum(11[,ULCD])
- [GUIMouse](#) - sum(12,startX,startY,dataStrN)
- [GUIMenu](#) - sum(13,X,Y,"icon","title","itemlist","labellist")
- [PushAnsStack](#) - sum(14[,stackNumber])
- [PopAnsStack](#) - sum(15[,stackNumber])
- [ClearAnsStack](#) - sum(16[,stackNumber])

Celtic Routines

- [ToggleSet](#) - det(0,"FN",f#) 0=Arc;1=Lk;2=PA;3=Hid;4=Del;5=Crt;6=PS;7=Stat
- [NumToString](#) - det(1,var) Out:type code
- [GetListElem](#) - det(2,"LN",elem) out:lname(elem) -> Real/Cpx
- [GetArgType](#) - det(3,argofanytype) filetype code
- [ChkStats](#) - det(4,f#) 0=RAM 1=ARC 2=ID 3=OSVER
- [LineRead](#) - det(5,"FN",line#,[#oflines])
- [LineWrite](#) - det(6,"STR","FN",line#)
- [LineErase](#) - det(7,"FN",line#,[#oflines])
- [LineReplace](#) - det(8,"STR","FN",line#,[#oflines])
- [FindProg](#) - det(9,["SEARCH"])
- [UngroupFile](#) - det(10,"GFN")
- [GetGroup](#) - det(11,"GFN")
- [ExtGroup](#) - det(12,"GFN",item#)
- [GroupMem](#) - det(13,"GFN",item#)
- [BinRead](#) - det(14,"FN",bytestart,#ofbytes)
- [BinWrite](#) - det(15,"HEX","FN",bytestart)
- [BinDelete](#) - det(16,"FN",bytestart,del#ofbytes)
- [HexToBin](#) - det(17,"484558")
- [BinToHex](#) - det(18,"BIN")
- [FastCopy](#) - det(19)
- [ExecHex](#) - det(20,"C9")
- [Edit1Byte](#) - det(21,Str?,StartByte,ReplaceWithThisByte)
- [ModeChange](#) - det(22...) Not implemented
- [IndexFile](#) - det(23,"FILENAME","NEWINDEXNAME")
- [LookupIndex](#) - det(24,"FILENAME","INDEXNAME",line#,[#oflines])
- [ErrorHandle](#) - det(25,function,string)
- [MatToStr](#) - det(26,Matrix#)
- [StringRead](#) - det(27,"binstring",start,readThisMany)
- [HexToDec](#) - det(28,"HEXSTRING")
- [DecToHex](#) - det(29,SomeRealNumber,[autoOverride])

Doors CS 7 SDK

- [EditWord](#) - det(30,start_byte,replace_with_this_word)
- [BitOperate](#) - det(31,value1,value2,logic)
- [GetProgList](#) - det(32,"SEARCHSTRING",[type])

PicArc Routines

- [DBQuery1](#) - identity(0,"DB",f#,args) 0=fm;1=ad;2=dl;3=di;4=ac;5=cb;6=cp
- [TogglePic](#) - identity(1,pic#,f#) 0-7=picToBuf;8=newPic;9=TARC;10=del
- [GetPicGroup](#) - identity(2,"GFN")
- [ExtPicGroup](#) - identity(3,"GFN",pic#)
- [StringTile](#) - identity(4,"BSTR",xS,xE,yS,yE,W,xO,yO,pic#S,displagic)
- [PutSprite](#) - identity(5,ScrewDocumentingThisCommand)
- [ShiftScreen](#) - identity(6,sft#,sftDir,scrnUpdate)
- [DrawBox](#) - identity(7,x1,y1,x2,y2,dispmeth)
- [FillMap](#) - identity(8,"HEXSTRING",right,down,LOGIC,updateLCD)
- [BoxShift](#) - identity(9,x,y,w,h,shiftType,shiftRepeat,updateLCD)
- [DrawText](#) - identity(10,flag,[x,y],"TEXT")
- [CopyBox](#) - identity(xx,x1s,y1s,x2s,y2s,x1d,y1d,x2d,y2d,dispmeth)

XLib Routines

- [ClearScreen](#) - real(0,ULCD)
- [DrawSprite](#) - real(1,x,y,w,h,p#,pX,pY,logic,flp,ULCD)
- [DrawTileMap](#) - real(2,MTRX#,xO,yO,W,H,xS,xE,yS,yE,p#,LOGIC,tSiz,ULCD)
- [RecallPic](#) - real(3,p#,LOGIC,ULCD)
- [ScrollScreen](#) - real(4,scrDir,scrStep,ULCD)
- [ChngConstrast](#) - real(5,f#,val). 0=SetContrast;1=ReturnCurrentContrast
- [UpdateLCD](#) - real(6)
- [RunIndicator](#) - real(7,value) 0=off;1=on
- [GetKey](#) - real(8) Returns keycode
- [CreatePIC](#) - real(9,f#,p#,[1=64 rows]) 0=stoPic;1=DelPic
- [ExecArcPrgm](#) - real(10,f#,tPrg#) Pass Ans="FN"
- [GetCalcVer](#) - real(11) Returns calcver. 0=83+;1=SE;2=84+;3=SE
- [DrawShape](#) - real(12,shapeType,x1,y1,x2,y2,ULCD)
- [TextMode](#) - real(13,f#,[chr]) 0=inv;1=un;2=lwrcse;3=un;4=outputchar
- [CheckRAM](#) - real(14) Returns amount of free RAM.

Omnicalc Routines

- [sprite](#) - real(20,pic#,picX,picY,w,h,x,y,[logic])
- [ExecAsm](#) - real(33,"HEXSTRING")

DCSB Lib Routine Details

StringWidth

<http://dcs.cemetech.net/index.php?title=BasicLibs:StringWidth>

Description

This routine will find the width in pixels of any TI-BASIC screen, if it is written on the graphscreen in small variable-width font. For example, it would return 1 for a single space, 14 for the "cos(" token, and 18 for "Hello". Since it does not actually draw the string, it accepts arbitrarily-long strings, including those wider than 96 pixels.

Example Usage

```
PROGRAM:SUM0TEST
:Input "STRING:",Str1
:Disp "WIDTH:",sum(0,Str1
```

Technical Details

Arguments

sum(0,"STRING"

Outputs

The width in pixels of "STRING" is returned in Ans from this routine, and can be used as Ans or stored in a variable.

Destroyed

Ans

ScanForPx

<http://dcs.cemetech.net/index.php?title=BasicLibs:ScanForPx>

Description

This routine scans the graphscreen in a given direction from a given starting point, and returns the coordinate of the last pixel it encounters that is the same color as the starting pixel. For example, if the starting pixel is black, and 2 (down) is specified as the direction, the routine will move down one pixel at a time from the starting pixel until it finds a white pixel, then return the y-coordinate of the pixel **above** that pixel. If the pixel directly below the starting pixel was the first white pixel, it would return the coordinates of the starting pixel. If the edge of the screen is reached, the coordinate of the edge pixel will be returned (0 for up or left, 95 for right, 63 for down). This routine is useful for things like flood filling and collision detection.

Technical Details

Arguments

sum(1,X,Y,DIR

X: x-coordinate of starting pixel

Y: y-coordinate of starting pixel

DIR: Direction to scan. 1=up, 2=down, 3=left, 4=right, all others invalid

Outputs

The coordinate of the last pixel of the same color as the starting pixel in the given direction is returned in Ans. If the chosen direction is up or down, the coordinate is a y-coordinate (the x will be the same as the initial x). If the direction is left or right, Ans will contain an x-coordinate.

Destroyed

Ans

HomescreenToGraphscreen

<http://dcs.cemetech.net/index.php?title=BasicLibs:HomescreenToGraphscreen>

Description

Simply copy the homescreen to the graphscreen. Because of the limited metadata that the TI-OS stores about homescreen characters, this can only apply formatting like text inversion to the entire screen or none of it. This routine can optionally update the LCD with the contents of the graph screen.

Technical Details**Arguments**

sum(2,ULCD

ULCD: 1 to update the LCD with the new graph screen contents, 0 to leave it as-is

Outputs

Graph buffer updated.

Destroyed

None

EnumeratePicVars

<http://dcs.cemetech.net/index.php?title=BasicLibs:EnumeratePicVars>

Description

Enumerate all picture variables present on the calculator, between Pic 1 and Pic 255 (Pic 0 is indicated as #10). This function returns a list containing numbers, each pertaining to a picture variable present. A negative number indicates that the particular picture is archived. If this function returns a one-element list containing the number 0, then no pictures are present.

Technical Details

Arguments

sum(3)

Outputs

A list is returned in Ans containing as many elements as there are Pic variables present. The numerical value of each element indicates the presence of one Pic variable; a negative element indicates an archived Pic. Note that the list returned is NOT sorted. The case of no Pics present causes the list {0} to be returned.

Destroyed

Ans

ArcUnarcPic

<http://dcs.cemetech.net/index.php?title=BasicLibs:ArcUnarcPic>

Description

This routine will archive or unarchive a specified Pic variable. If the picture is already in the desired state, nothing happens. If the picture does not exist, nothing happens. This works with Pic 1 through Pic 255.

Technical Details

Arguments

sum(4,PIC,DESIRED

PIC: The number of the Pic variable to use. 3=Pic 3, 60=Pic 60, 10=Pic 0, for example

DESIRED: 0 if it should be put into (or left in) RAM, 1 if it should be put into (or left in) the Archive.

Outputs

The Picture is archived, left in RAM, or left undisturbed, depending on where it started and what the desired final state was. Ans is not modified.

Destroyed

None

DCSLibVersion

<http://dcs.cemetech.net/index.php?title=BasicLibs:DCSLibVersion>

Description

This routine simple returns an integer indicating which version of the Doors CS libraries are present. Note: this routine cannot be used to verify whether Doors CS libraries or present or not. For that, det([[42 must be used; the beginning of this chapter for details.

Technical Details

Arguments

sum(5)

Outputs

An integer is returned in Ans indicating the version of the DCSB Libs present. Known versions:

- 1: Supplies sum(0) through sum(12), as present in Doors CS 6.7 beta.
- 2: Supplies sum(0) through sum(13), as present in Doors CS 6.8 beta.
- 3: Supplies sum(0) through sum(16), as present in Doors CS 6.9 beta.

Destroyed

Ans

SimpleGUIMouse

<http://dcs.cemetech.net/index.php?title=BasicLibs:SimpleGUIMouse>

Description

This routine will invoke a simplified version of the Doors CS GUI Mouse. It runs until a click is registered, and will return regardless of where the click is. It does not take into account the contents of the screen or GUI stack, if any. It will properly preserve the graphscreen, and return with it in the same state as when it started.

Sample Usage

```

PROGRAM: SUM6TEST
: {45, 32, 1
: While 2≠Ans(3
: sum(6, Ans(1), Ans(2)
: Text(1, 1, "YOU CLICKED AT (", Ans(1), ", ", Ans(2), ") [", Ans(3), "]
: End

```

Technical Details

Arguments

sum(6,X,Y

X: The x-coordinate at which to start the mouse. Valid values are 0 through 95, inclusive.

Y: The y-coordinate at which to start the mouse. Valid values are 0 through 63, inclusive.

Outputs

This routine returns a 3-element list in the form {X,Y,CLICK}. X and Y are the final x-coordinate and y-coordinate of the mouse when a click was registered. The CLICK element of the list is 1 if the click was a left-click (2nd, TRACE, or ENTER) or a right-click (ALPHA or GRAPH).

Destroyed

Ans

PushGUIStack

<http://dcs.cemetech.net/index.php?title=BasicLibs:PushGUIStack>

Description

This routine pushes one element onto the GUI stack; it is among the most complex of the DCSB Libs. Read carefully the documentation for each function, as invalid or improper arguments may cause unpredictable behavior up to freezes and RAM clears. Please see the GUI API guide for assembly programmers for full information on how the GUI stack works. You can also check out the ASM equivalents of the DCSB Lib GUI stack routines at the GUI Tools page. The ASM equivalent of this function is [PushGUIStack](#).

Technical Details

Arguments

sum(7,TYPE,[ARGS...])

TYPE: The type of element to push onto the GUI stack. There are 24 total valid types, as detailed below. Each is linked to the ASM SDK page on that type.

- **[GUIRnull](#)**: sum(7,0,N). This is one of the three types that must be the first element pushed. The value of N should be 255 if the graphscreen should be completely erased before rendering the elements on top of this, or 254 if the contents of the graphscreen should be preserved.
- **[GUIRLargeWin](#)**: sum(7,1,ICON,TITLE). This is one of the three types that must be the first element pushed. ICON must be a 10-character hex string defining five bytes containing a 5x5 icon. The TITLE string may contain any typeable characters and tokens.
- **[GUIRSmallWin](#)**: sum(7,2,X,Y,ICON,TITLE). This is one of the three types that must be the first element pushed. X and Y are the coordinates of the top-left corner of the small window, and must be between 0 and 15, inclusive. An X or Y value greater than 15 may cause memory corruption and crashes. ICON must be a 10-character hex string defining five bytes containing a 5x5 icon. The TITLE string may contain any typeable characters and tokens.
- **[GUIRFullScreenImg](#)**: sum(7,3,PIC#). This will insert a fullscreen image into the GUI stack, and is usually used directly after sum(7,0,255), creation of a new GUI stack group headed by a GUIRnull. The value of Pic# specifies the Pic variable from which the image should be copied. The Pic must be in RAM, and as with all Pic functions, 10=Pic 0.
- **[GUIRText](#)**: sum(7,4,X,Y,0,"STRING"). The specified string will be displayed in the GUI stack, where X,Y is the top-left corner of the text. The 0 specifies what font to use; as of Doors CS 7.0, only one font is available, but future versions of Doors CS may add additional fonts. The STRING should not overflow the right edge of the container or the screen.
- **[GUIRWinButtons](#)**: sum(7,5,N). Display one or more of the possible window buttons at the top of this container. This should only be used if a GUIRSmallWin or GUIRLargeWin is the

head element of this GUI stack group. N is a sum of the possible values 32 (for the close or X button), 64 (for the maximize or square button), and 128 (for the minimize or underbar button). For example, to render the close and minimize buttons, N should equal $32+128 = 160$.

- **GUIRWrappedText**: `sum(7,6,X,Y,WIDTH,0,STRING)`. Draws a block of text starting at (X,Y), of width WIDTH, automatically wrapped as necessary. The WIDTH should not be wider than the container, nor should it cause any part of the text to be rendered below the bottom of the screen. The 0 indicates what font to use; as of Doors CS 7.0, only font 0 is available. Do not confuse this with the `GUIRTextMultiline` element.
- **GUIRButtonText**: `sum(7,7,X,Y,"STRING")`. Renders a clickable button at (X,Y) containing string STRING. If the button is clicked, a GUI Mouse call will be terminated.
- **GUIRButtonImg**: `sum(7,8,X,Y,IMWIDTH,BUTTONIMG,ICON)`. Renders a clickable button at (X,Y) containing the image specified by the hex code in ICON. The IMWIDTH may be 1 for 8 pixels, 2, for 16 pixels, etc. The BUTTONIMG specifies the width of the image in pixels, for example 5, 6, or 8 if IMWIDTH=1. The ICON is a string of hex nibbles 10*IMWIDTH characters long.
- **GUIRTextLineIn**: `sum(7,9,X,Y,WIDTH,MAXCHARS,STRING)`. Renders an interactive single-line text box WIDTH pixels wide. Up to MAXCHARS characters may be typed inside. The initial contents of the text box will be STRING, or blank if STRING is "".
- **GUIRRadio**: `sum(7,10,X,Y,GROUPID,STATE,STRING)`. Renders a radio button at (X,Y), including a descriptive STRING to the right of the button. The STATE argument can be 0 or 1, indicating whether the button should be initially unselected or selected, respectively. The GROUPID indicates which radio buttons are in the same group; when one radio button with a given GROUPID is clicked, all other radio buttons with the same GROUPID are unselected.
- **GUIRCheckbox**: `sum(7,11,X,Y,GROUPID,STATE,STRING)`. Renders a checkbox at (X,Y), including a descriptive STRING to the right of the box. The STATE argument can be 0 or 1, indicating whether the box should be initially unchecked or checked, respectively. The GROUPID indicates which check boxes are in the same group, but has no effect currently. It is included mainly for symmetry with `GUIRRadio` elements.
- **GUIRByteInt**: `sum(7,12,X,Y,INITIAL,MIN,MAX)`. A numerical input box, accepting a number between 0 and 255, inclusive. The MIN and MAX arguments specify some subset of that range as valid for this box. The INITIAL argument specifies what number it should start at when rendered.
- **GUIRWordInt**: `sum(7,13,X,Y,INITIAL,MIN,MAX)`. A numerical input box, similar to `GUIRByteInt` but accepting a number between 0 and 65535, inclusive. The MIN and MAX arguments specify some subset of that range as valid for this box. The INITIAL argument specifies what number it should start at when rendered.

- **GUIRHotspot**: `sum(7,14,X,Y,WIDTH,HEIGHT)`. An invisible clickable hotspot with upper-left corner at (X,Y) and a given WIDTH and HEIGHT. It masks any elements underneath, so a GUIRHotspot can be used to provide custom clickability on top of other elements.
- **GUIRTextMultiline**: `sum(7,15,X,Y,ROWS,WIDTH,STRING)`. An interactive multiline textbox with ROWS rows (ie, 6*ROWS pixels tall) and WIDTH pixels wide. The initial contents of the textbox is STRING; line returns can be represented with the Stat Plots symbol "dot". An empty initial STRING "" is also acceptable.
- **GUIRSprite**: `sum(7,16,X,Y,HEIGHT,SPRITE)`. An 8-pixel wide, HEIGHT-pixel-high sprite. SPRITE is a 2*HEIGHT-character string of hex nibbles. (X,Y) is the upper-left corner of the sprite.
- **GUIRLargeSprite**: `sum(7,17,X,Y,BYTEWIDTH,PXHEIGHT,SPRITE)`. An 8*BYTEWIDTH-pixel-wide, PXHEIGHT-pixel-high sprite. X should be less than 97-8*BYTEWIDTH, and Y should be less than 65-PXHEIGHT. SPRITE should be a string of hex nibbles, 2*PXHEIGHT*BYTEWIDTH characters long.
- **GUIRPassIn**: `sum(7,18,X,Y,WIDTH,MAXCHARS,STRING)`. Renders an interactive single-line password input box WIDTH pixels wide. Up to MAXCHARS characters may be typed inside. The initial contents of the text box will be STRING, or blank if STRING is "". The functionality of this element is identical to that of GUIRTextLineIn except that the characters are obfuscated.
- **GUIRScrollVert**: `sum(7,19,X,Y,HEIGHT,ID,INC/DEC,MIN,MAX,CUR)`. Render a vertical scrollbar starting at upper-left at (X,Y), with scroll area height HEIGHT (total vertical height 12+HEIGHT). The MIN, MAX, and CUR values define the size of the scroll slider and its current position. The ID is arbitrary, but should be unique for future use. INC/DEC is also irrelevant in the context of the DCSBLibs, and should be left as 1.
- **GUIRScrollHoriz**: `sum(7,20,X,Y,WIDTH,ID,INC/DEC,MIN,MAX,CUR)`. Render a horizontal scrollbar starting at upper-left at (X,Y), with scroll area width WIDTH (total horizontal width 12+WIDTH). The MIN, MAX, and CUR values define the size of the scroll slider and its current position. The ID is arbitrary, but should be unique for future use. INC/DEC is also irrelevant in the context of the DCSBLibs, and should be left as 1.
- **GUIRBorder**: `sum(7,21,X,Y,WIDTH,HEIGHT,COLOR)`. Render an outlined, unfilled rectangle with upper-left corner (X,Y), width WIDTH, and height HEIGHT. COLOR is 0=white, 1=black.
- **GUIRRect**: `sum(7,22,X,Y,WIDTH,HEIGHT,COLOR)`. Render a filled rectangle with upper-left corner (X,Y), width WIDTH, and height HEIGHT. COLOR is 0=white, 1=black, 2=invert.
- *Note the **absence of a `sum(7,23,...)` function.*** *GUIRCustom is not a valid DCSB Libs item.*
- **GUIRMouseCursor**: `sum(7,24,X1*256+Y1,X2*256+Y2,MASK,CURSOR)`. (X1,Y1) is the top-left corner of the area; (X2,Y2) is the bottom-right corner. Note that the coordinates are LCD-based, not based on the coordinates of the current window. The MASK is a 16-character string of hex nibbles specifying an 8x8 white-on-black mask, where black is transparent and white is opaque. The CURSOR is a similar 16-character string of hex

nibbles specifying an 8x8 sprite. Inside the specified area, the mouse cursor will be changed from the default to the specified cursor.

Outputs

The GUI stack is modified to include the new element, and enlarged to hold it. Note that memory constraints apply; it is recommended that the programmer be cautious and verify there is enough available memory before large pushes to the stack.

Destroyed

None

PopGUIStacks

<http://dcs.cemetech.net/index.php?title=BasicLibs:PopGUIStacks>

Description

This routine pops one or more elements off of the GUI stack, as pushed by `sum(7,...)`. Be very careful not to pop more elements than you have pushed, as this may cause unpredictable behavior up to freezes and RAM clears! You may pop off two elements and then push one element, for example, but make sure the total number of elements does not fall below 0 at any time. If you want to remove all elements on the GUI stack, you may instead use `CloseGUIStack`, or `sum(10)`. Please see the GUI API guide for assembly programmers for full information on how the GUI stack works. You can also check out the ASM equivalents of the DCSB Lib GUI stack routines at the GUI Tools page. The ASM equivalent of this function is [PopGUIStacks](#).

Technical Details**Arguments**

`sum(8,ELEMENTS`

`ELEMENTS`: The number of elements to pop off of the GUI stack.

Outputs

None

Destroyed

None

OpenGUIStack

<http://dcs.cemetech.net/index.php?title=BasicLibs:OpenGUIStack>

Description

This routine opens the GUI stack for use by your BASIC program. Please see the GUI API guide for assembly programmers for full information on how the GUI stack works. You can also check out the ASM equivalents of the DCSB Lib GUI stack routines at the GUI Tools page. The

ASM equivalent of this function is [OpenGUIStack](#). Please note that if you [PushGUIStack](#) without calling [OpenGUIStack](#) first, Doors CS will automatically open the GUI stack.

Note: It is vital that your program be polite and close the GUI stack via `sum(10)`, [CloseGUIStack](#), before Return'ing or Stop'ing!

Technical Details

Arguments

`sum(9`

Outputs

None

Destroyed

None

CloseGUIStack

<http://dcs.cemetech.net/index.php?title=BasicLibs:CloseGUIStack>

Description

This routine closes the GUI stack, removing all elements in it and freeing the memory used by the GUI stack Appvar. If you call this function without first having called [OpenGUIStack](#) or [PushGUIStack](#), it does nothing. Please see the GUI API guide for assembly programmers for full information on how the GUI stack works. You can also check out the ASM equivalents of the DCSB Lib GUI stack routines at the GUI Tools page. The ASM equivalent of this function is [CloseGUIStack](#).

Technical Details

Arguments

`sum(10`

Outputs

None

Destroyed

None

RenderGUI

<http://dcs.cemetech.net/index.php?title=BasicLibs:RenderGUI>

Description

This routine renders the contents of the GUI stack, but then returns control to the calling BASIC program rather than invoking the [GUIMouse](#). This routine is useful if you wish to render

an interface via the DCS GUI stack system, but either don't wish to use the mouse, or wish to use the [SimpleGUIMouse](#) instead of the full GUI mouse. Please see the [GUI API chapter](#) for assembly programmers for full information on how the GUI stack works. You can also check out the ASM equivalents of the DCSB Lib GUI stack routines in the same chapter. The ASM equivalent of this function is [RenderGUI](#).

Technical Details

Arguments

sum(11[,ULCD])

If the function is called as sum(11), the GUI is rendered to the graph buffer and copied to the LCD. Sum(11,1) will also achieve the same thing. However, sum(11,0) will render the GUI to the graph buffer but not copy it to the LCD.

Outputs

None

Destroyed

None

GUIMouse

<http://dcs.cemetech.net/index.php?title=BasicLibs:GUIMouse>

Description

This routine both renders the current GUI stack, and invokes the GUI mouse; it is not necessary to call sum(11), RenderGUI, before this function. Please see the GUI API guide for assembly programmers for full information on how the GUI stack works. You can also check out the ASM equivalents of the DCSB Lib GUI stack routines at the GUI Tools page. The ASM equivalent of this function is ASMLibs:GUIMouse.

To avoid possible situations where a BASIC programmer calls sum(12) without any way of escaping out of the call, like a GUIRButton or GUIRWinButtons, [STO>] acts as a panic buttons, and will return with the special outputs Ans(3)=0 and Ans(4)={number of interactive hotspots in stack}. If you determine what was clicked by checking Ans(4), you can either handle this as a special case or ignore it and re-loop, giving you a short time to press [ON] and trigger an ERR:BREAK. Savvy programmers may even find a way to use this as a feature.

Notes: Do not call this function without at least two elements on the GUI stack, namely a GUI master (specifically, a GUIRnull, a GUIRSmallWin, or a GUIRLargeWin), and a hotspot, button, or GUIRWinButtons that will allow control to return to the BASIC program. If you call this function without anything clickable in the GUI stack, then there will be no way to break out of the mouse other than [STO>]. The [ON] key turns off the calculator when the GUI mouse is

active. The calculator will also APD if the GUI mouse is active and idle for two to three minutes.

Technical Details

Arguments

sum(12,X,Y,N

X: Initial x-coordinate of mouse.

Y: Initial y-coordinate of mouse.

N: String in which to store interactive element contents. 4=Str 4, 8=Str 8, 10=Str 0, for example.

Outputs

This routine has two output variables:

- **Ans:** A four-element list is returned in Ans containing {X,Y,CLICK,TRIGGER}. The X and Y values are the final x- and y-coordinates of the mouse when a click occurred. The CLICK element is 1 if a left click was registered, or 2 for a right click. The value of TRIGGER indicates which hotspot in the GUI stack was clicked; its meaning is very dependent on the contents you have pushed into the GUI stack. If you pushed a GUIRWinButtons containing only an X and a GUIRButtonText, for instance, clicking the X will return TRIGGER=0, and clicking the text button will return TRIGGER=1. Scrollbars have two hotspots, so clicking the left or up arrow will return a distinct trigger from clicking the right or down arrow. You may also wish to correlate the values of {X,Y} and TRIGGER for more fine-grained detection. Generally, trial-and-error is the best (and most intuitive) way to figure out what TRIGGER values correspond to what GUI element for a particular GUI stack configuration.
- **Str N:** The contents of most interactive GUI elements are returned concatenated into a single string, Str N, where the value of N is specified as one of the arguments to this function. Elements are separated with the Stat Plots "+" symbol (different from the addition symbol). Line returns in string elements are represented by the Stat Plots "dot" symbol. This string will always contain the same number of Stat Plots "plus" symbols as it does elements, preceeding each element by its "plus" symbol. The inString() and sub() functions, combined when necessary with the expr() command, can be used to extract all the individual string and numbers from this string.

Destroyed

Ans, StrN (see above)

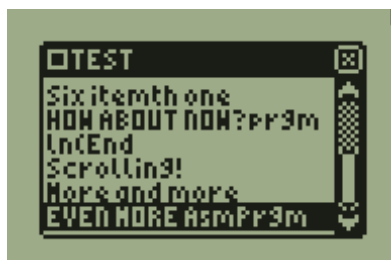
GUIMenu

<http://dcs.cemetech.net/index.php?title=BasicLibs:GUIMenu>

Description

This routine offers a very simple way to invoke a fully-GUIed alternative to the TI-BASIC Menu(command. The user can specify an icon, window location, title, and list of items and labels. Just like Menu(, the sum(13) command will Goto a specific label corresponding to the item selected, or return an error if no such label can be found. An alternate mode has been added in which if the LABELLIST is omitted, then the index of the chosen menu item will instead be returned in Ans.

Sample Usage



```
:sum(13,8,8,"TEST","F8888888F8","Item 1+ITEM
TWO+tan(*tan(4+log(^2X+][+^2^-../+Six itemth one+HOW ABOUT
NOW?prgm+ln(End+Scrolling!+More and more+EVEN MORE AsmPrgm+", "AAABACAD
E EABACAD EAB
:Lbl AA:ClrHome
:Pause "1
:Return
:Lbl AB:ClrHome
:Pause "2
:Return
:Lbl AC:ClrHome
:Pause "3
:Return
:Lbl AD:ClrHome
:Pause "4
:Return
:Lbl E:ClrHome
:Pause "5
:Return
```

Technical Details

Please note that the weird + symbol is the STATS Plus, **not** the normal plus key.

Arguments

sum(13,X,Y,TITLE,ICON,ITEMLIST,LABELLIST

X: X-coordinate of top-left corner of small window

Y: Y-coordinate of top-left corner of small window

ICON: 10-character hex string defining a 5x5 icon

TITLE: A string to be used as the window title

ITEMLIST: A list of items to be in the menu. Each menu item should conclude with the Stats Plots "+" symbol (in [2nd][Y=][>][>][2]), so the ITEMLIST string should contain as many Stats Plots "+" symbols as items.

LABELLIST: A string of 2*(number of items) characters. Two-character labels, like AA or Q2 or PA, are represented verbatim. One-character labels such as A or U are represented as a two character string, where the first character is a space, such as " A" or " U". Take a look at label E in the sample below for another example. If this argument is omitted completely, then no Goto'ing will be attempted; instead, Doors CS will simply return the index of the chosen menu item.

Outputs

If the LABELLIST is specified, this routine will not modify Ans, but will Goto to the label corresponding to the chosen menu item. If the LABELLIST is instead omitted, Ans will indicate which item was chosen.

Destroyed

Contents of LCD/graph buffer

PushAnsStack

<http://dcs.cemetech.net/index.php?title=BasicLibs:PushAnsStack>

Description

This routine pushes the current value of Ans into a special Ans stack. The PopAnsStack command is the reverse function, popping a value back into Ans. It is recommended that you call ClearAnsStack before touching the AnsStack in case another programmer failed to clean it up properly.

If no second argument is provided, then the value in Ans is pushed to the default stack, called Ans0. However, a second argument between 0 and 9 inclusive will instead specify that the value should be pushed to one of Ans0 through Ans9. PopAnsStack allows a similar second argument.

Technical Details

Arguments

sum(14[,stackNumber])

Outputs

The value of Ans is pushed to the AnsStack if and only if there is enough memory, and the value in Ans is one of the following types:

- Real Number
- Complex Number
- Real List

Doors CS 7 SDK

- Complex List
- String
- Real Matrix

If there is insufficient error, an ERR:MEM will be thrown. If the value in Ans is not one of the acceptable types, no item is pushed to the stack, but execution continues.

Destroyed

None

PopAnsStack

<http://dcs.cemetech.net/index.php?title=BasicLibs:PopAnsStack>

Description

This routine pops the current value of Ans from a special Ans stack. The PushAnsStack command is the reverse function, pushing a value into Ans. It is recommended that you call ClearAnsStack before touching the AnsStack in case another programmer failed to clean it up properly.

If no second argument is provided, then the value in Ans is popped from the default stack, called Ans0. However, a second argument between 0 and 9 inclusive will instead specify that the value should be popped from one of Ans0 through Ans9. PushAnsStack allows a similar second argument.

Technical Details**Arguments**

sum(15,[stackNumber])

Outputs

The value of Ans is popped from the AnsStack if and only if there is enough memory, and the value in Ans is one of the following types:

- Real Number
- Complex Number
- Real List
- Complex List
- String
- Real Matrix

If the AnsStack is empty, Ans is left untouched.

Destroyed

Ans

ClearAnsStack

<http://dcs.cemetech.net/index.php?title=BasicLibs:ClearAnsStack>

Description

This routine clears the special Doors CS AnsStack. The PushAnsStack and PopAnsStack functions are used to add and remove items from the AnsStack. If no second argument is specified, then all AnsStacks, Ans0 through Ans9, are cleared and deleted. Specify sum(16,0) to clear the default AnsStack, Ans0. Specify another value to clear and delete that AnsStack. Valid values are 0-9.

Technical Details

Arguments

sum(16,[stackNumber])

Outputs

AnsStack cleared and deleted, if it existed.

Destroyed

None

Celtic III Library Routine Details

ToggleSet

<http://dcs.cemetech.net/index.php?title=BasicLibs:ToggleSet>

Description

Performs an action with a file given the function_number provided. If function_number is...

- 0 = Archive/unarchive a file. The output will be zero if it has been unarchived. Some other number if archived.
- 1 = Lock/unlock a file. The output is 0 if unlocked, 1 if it is locked, 2 if you attempted this on an appvar.
- 2 = Program to appvar and vice versa. A "1" will be output if the operation was a success.
- 3 = Hide/unhide a file. Outputs 0 if the file is visible, 1 if it is now hidden.
- 4 = Delete. Just. Deletes a file. Not undoable. Be careful on how you use this.
- 5 = Create. Creates a zero-sized file with the name you choose. There so you can insert your own stuff in it.
- 6 = Prog to string. Outputs the entire program file into a string so you can paste it into some other file.
- 7 = Output stats. Somewhat complicated. It outputs three letters followed by five digits indicating its size. Those three letters will tell you a lot about the file:
 - 1st letter: A= archived, R= in RAM
 - 2nd letter: H= hidden, V= visible
 - 3rd letter: W= writable, L= locked (uneditable)

Naturally, archived files are uneditable. To get just the size of the file, do this:

```
expr ( sub ( det ( 0 , "FILENAME" , 7 ) , 4 , 5 ) )
```

If you omit the function number, the command defaults to the archive/unarchive function. Useful for saving bytes.

Technical Details

Arguments

`det(0,"FILENAME",[function_number])`

Outputs

(See Description)

NumToString

<http://dcs.cemetech.net/index.php?title=BasicLibs:NumToString>

Description

Converts any variable of the real type to a string. This includes real vars, real lists, and matrices. Useful if you want to store variables into program files, perhaps as a high score or just data storage for later use. Note that storing numbers this way takes up a whole lot less space unless your numbers are sporting many decimals.

If your input is a list and it is in quotes, you'll also be able to read that list even if it's from archive. For rules about inputting a list, see the GetListElem command (det(2)).

Technical Details**Arguments**

det(1,variable_of_any_real_type)

Outputs

(See description)

GetListElem

<http://dcs.cemetech.net/index.php?title=BasicLibs:GetListElem>

Description

Gets an element from a list. If you could use the standard TI-OS function on it, you should, but this command is very useful if the list is ARCHIVED. Otherwise, it works just like your standard L1(n) command. If you omit the list element, or set it to zero, this function will return the dimensions of the list.

RULES: If the name is predefined (L1 through L6), just put that in quotes. If it is a user-defined list, you must prefix the name with that little "L". This is done automatically if you paste the name of the list from the list variable list.

Technical Details**Arguments**

det(2,"LISTNAME",[get_this_element])

Outputs

Specified element of the specified list

GetArgType

<http://dcs.cemetech.net/index.php?title=BasicLibs:GetArgType>

Description

Outputs a real # depicting the type of variable that was input. A table follows of the possible non-erroneous input that can be made. Useful for determining whether or not an output to a

Celtic III command was a number or an error. To prevent this from altering the Ans variable, use this command within if() statements and the like.

- Real = 0
- List = 1
- Matrix = 2
- Equation = 3
- String = 4
- Complex = 12
- Cpx List = 13

Technical Details

Arguments

det(3,argument_of_any_type)

Outputs

Real number indicating type of argument (see description).

ChkStats

<http://dcs.cemetech.net/index.php?title=BasicLibs:ChkStats>

Description

This is a multi-purpose command used to read some of the system's status. Just provide a function, and an output will result.

Technical Details

Arguments

det(4,function)

Possible functions:

- 0 = Outputs value of free RAM
- 1 = Outputs value of free ROM (Archive)
- 2 = Hex string output of first ten digits of calc serial (still not quite enough to identify each calc)
- 3 = Get calc's OS version (string, such as "1.13") (note that the period will be bugged. It still works)

Outputs

A number or string.

LineRead

<http://dcs.cemetech.net/index.php?title=BasicLibs:LineRead>

Description

Reads a file (BASIC program) starting at the specified line and, if included, reads that many more lines. If a number of lines is included, it adds that many more to the output. For example, if you wanted to read from the first line and grab one more line after that from the file "FOO", you'd have the following:

```
det ( 5 , " FOO " , 1 , 1 )
```

If you just want to find how many lines there are in the file, just try to read the 0th line. It'll output a real number containing the number of lines in the file.

Technical Details

Arguments

```
det(5,"FILENAME",read_this_line,[by_this_many_more_lines])
```

Outputs

(See description)

LineWrite

<http://dcs.cemetech.net/index.php?title=BasicLibs:LineWrite>

Description

Writes the contents of a string to a file starting at the specified line. A newline character is appended to the end of the string you want to write, so you do not need to include it.

Technical Details

Arguments

```
det(6,"FILENAME","STRINGTOWRITE",start_at_this_line)
```

Outputs

(See description)

LineErase

<http://dcs.cemetech.net/index.php?title=BasicLibs:LineErase>

Description

Deletes a line from a file starting at the specified line and if specified, deletes that many more lines from the file. This is NOT an undoable operation, so be careful.

Technical Details

Arguments

```
det(7,"FILENAME",delete_this_line,[and_these_many_more])
```

Outputs

Deletes lines from a specified file.

LineReplace

<http://dcs.cemetech.net/index.php?title=BasicLibs:LineReplace>

Description

Replaces the specified line at start with the contents of your string. If you specified a continue, it would truncate that many more lines. For example, if you wanted to replace 4 lines with just the word "BAR" from the second line from the file "FOO":

```
det ( 8 , " BAR " , " FOO " , 2 , 3 )
```

Technical Details**Arguments**

```
det(8,"STRING","FILENAME",line_number,[number_of_lines])
```

Outputs

Replaces line(s) in specified file

FindProg

<http://dcs.cemetech.net/index.php?title=BasicLibs:FindProg>

Description

The search phrase is NOT part of the filename. It is, instead, the leading characters in the file you want to search for. The output will be a space-delimited (separated by spaces) string containing the names of the files that match the search phrase. For example, say you had a few programs that start with ":CCN". You would get all files that would match that if you did:

```
det ( 9 , " :CCN " )
```

If you wanted to get a complete program/appvar file listing then you'd just omit the search phrase. You will get some oddly-named program files as the first of the results, since they are system-reserved program files with illegal file names.

A neat little example with this command is locating all ION-type files. For that, you'd use an additional command inline with this one.

```
det ( 9 , det ( 17 , " BB6DC930 " ) )
```

That command uses the HEXTOBIN command to convert the hex to the binary that ASM programs (w/ION Head) are stored in.

Note: This command is not to be confused with GETPROGLIST, which outputs a string consisting of files whose NAMES start with the specified characters.

Technical Details

Arguments

det(9,["SEARCHPHRASE"])

Outputs

Finds specified file(s). See description.

UngroupFile

<http://dcs.cemetech.net/index.php?title=BasicLibs:UngroupFile>

Description

Fully ungroups files using an experimental routine designed to overcome the OS version @ | > 1.13 limit. Buggy, but should still work for program-type-only group files.

Default behavior is to skip any pre-existing files, but if [option] is set to 1, the behavior changes to overwrite. Careful when you do this because overwritten files are gone. Forever if in RAM and in archive if a garbage-collect was done over after the file went missing.

Technical Details

Arguments

det(10,"GROUPFILENAME",[option])

Outputs

Ungroups the specified file, if it exists.

GetGroup

<http://dcs.cemetech.net/index.php?title=BasicLibs:GetGroup>

Description

This generates a space-delimited string containing all the file names of programs and appvars that are found in the group in the order that they are found. The order is important, as it lends itself well to the ExtGroup command.

Technical Details

Arguments

det(11,"GROUPFILENAME")

Outputs

String specifying the contents of the given group file.

ExtGroup

<http://dcs.cemetech.net/index.php?title=BasicLibs:ExtGroup>

Description

Extracts the nth item from a group in the order it was found from the previous command. Outputs a real "1" if this command was successful.

Technical Details

Arguments

det(12,"GROUPFILENAME",extract_this_item)

Outputs

Ans = 1 if the command succeeded, otherwise it was unsuccessful (for example, specified an item larger than the number of items in the group) Extracts one specified file out of a group.

GroupMem

<http://dcs.cemetech.net/index.php?title=BasicLibs:GroupMem>

Description

Instead of extracting the nth item from a group, this determines how much data is in the file. Add some reasonable number, like 13, and you have the amount of memory this file would take up in RAM if you were to extract this. Great for group extraction managers.

Technical Details

Arguments

det(13,"GROUPFILENAME",get_size_of_this_item)

Outputs

Ans = the amount of memory consumed by the data portion of the specified file in the specified group.

BinRead

<http://dcs.cemetech.net/index.php?title=BasicLibs:BinRead>

Description

Reads a file starting at a specified byte and continues reading for the specified number of bytes. The output will be a hex string representing those bytes. The zeroth byte is the start of the file. Additional note: The starting point will accept a negative number but it'll only work correctly if the file is in RAM.

Technical Details

Arguments

```
det(14,"FILENAME",start_at_this_byte,read_this_many_bytes)
```

Outputs

Reads a given number of bytes from a specified program or file.

BinWrite

<http://dcs.cemetech.net/index.php?title=BasicLibs:BinWrite>

Description

Converts the hex string into bytes and then inserts that data into a file starting at the specified byte. The zeroth byte is the start of the file.

Technical Details**Arguments**

```
det(15,"FILENAME","HEXSTRING",start_at_this_byte)
```

Outputs

Modifies the specified file

BinDelete

<http://dcs.cemetech.net/index.php?title=BasicLibs:BinDelete>

Description

Deletes a number of bytes from a file starting at the specified location. The zeroth byte is the start of the file.

Technical Details**Arguments**

```
det(16,"FILENAME",start_here,delete_this_many_bytes)
```

Outputs

Deletes bytes from a file at a location.

HexToBin

<http://dcs.cemetech.net/index.php?title=BasicLibs:HexToBin>

Description

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

```
det(17,"41424344")
```

The result would be: "ABCD", since each of those hex digits equate to the tokens 'A','B','C', and 'D' in that order. Very useful if you want to provide input that would otherwise be impossible

from BASIC, such as the start of an ION shell header for the FINDPROG command, or supplying binary-only input for some of the few commands that require it.

This command replaces the TOKENCODE command, since using HEXTOBIN is more convenient due to the fact that most token equates are in hexadecimal, not decimal. You can also grab as many tokens as you want through this.

Technical Details

Arguments

det(17,"HEXSTRING")

Outputs

Converts hex code to binary.

BinToHex

<http://dcs.cemetech.net/index.php?title=BasicLibs:BinToHex>

Description

Just the opposite of the HEXTOBIN command. This converts a binary string back to hexadecimal. Useful for making sense of some ASM code/data files.

Technical Details

Arguments

det(18,"BINSTRING")

Outputs

Converts a binary string into a string of hex characters.

FastCopy

<http://dcs.cemetech.net/index.php?title=BasicLibs:ToggleSet>

Description

It calls the IFastCopy routine to update the LCD with the graph buffer.

Technical Details

Arguments

det(19)

Outputs

Copies the graph buffer to the LCD. Ans is unmodified.

ExecHex

<http://dcs.cemetech.net/index.php?title=BasicLibs:ExecHex>

Description

It converts up to 767 bytes into code (1534 hex characters) and executes it from the savesscreen buffer, useful for implementing simple stuff that was missed by a library as large as Celtic III/DCS7. Although it automatically appends a C9 (RET) instruction at the end of the string, you can still seriously screw up your calculator if used with erroneous ASM code.

Technical Details**Arguments**

det(20,"HEXSTRING")

Outputs

Executes a given string of hex as binary.

Edit1Byte

<http://dcs.cemetech.net/index.php?title=BasicLibs:Edit1Byte>

Description

Replaces a byte in some string variable, Str1 to Str0, using a decimal number 0 thru 255 in replace_with_this_byte within that string at this specified offset (at_this_byte), all of which start at zero. You will get an error if you try to overwrite something past the end of this string.

Since the specified string is edited directly, there will be no output and thus keep Ans intact, except if an error occurred. This means you cannot put in an immediate string as nothing but a discarded temporary variable will have been changed. Useful for quietly editing a string.

Technical Details**Arguments**

det(21,StringVar,at_this_byte,replace_with_this_byte)

Outputs

Replace a byte in a string.

ModeChange

<http://dcs.cemetech.net/index.php?title=BasicLibs:ModeChange>

Description

Allows one to read or change stuff in Celtic III's hidden appvar. Used for settings or reading status of delayed functions. The offset starts at 0.

If function= ...

- 0: Reads data at offset.
- 1: Writes [newvalue] to the offset. Old value is read out

More information about which offset does what will be explained later. Try not to pass bogus values into this command, as you CAN cause improper application function if used incorrectly.

Technical Details

Arguments

det(22,function,offset,[newvalue])

Outputs

Change Celtic modes.

IndexFile

<http://dcs.cemetech.net/index.php?title=BasicLibs:IndexFile>

Description

Creates an index of a program file and stores it as another program file as indicated by INDEXNAME. This is for use with the following command, LookupIndex

Any pre-existing files will be overwritten automatically, as if to update the index, so please take care that an important file is not specified. If the new index cannot be fit in memory after the old one is deleted, a new one is not created. Unfortunately, you will have lost the old index. Keep a tight rein on memory.

Technical Details

Arguments

det(23,"FILENAME","INDEXNAME")

Outputs

The index is created from FILENAME and stored in INDEXNAME, if enough memory exists.

LookupIndex

<http://dcs.cemetech.net/index.php?title=BasicLibs:LookupIndex>

Description

Uses the index file specified for quicker and much more consistent lookup of lines in a file. Very useful for large data files that would take a while to scan through using the normal lookup method.

The major drawback is that once a file is indexed, the file cannot be changed unless re-indexed by command 23. A good feature is that for purely read-only purposes, BOTH the program and index files can be archived.

Technical Details

Arguments

det(24,"FILENAME","INDEXNAME",this_line,[by_this_many])

Outputs

See Description

ErrorHandle

<http://dcs.cemetech.net/index.php?title=BasicLibs:ErrorHandle>

Description

Executes BASIC code with an error handler installed. That means if the code you execute can do anything it wants including divide by zero. The execution will end but an obvious system error will not trigger. Instead, this function will return with a value that indicates the error condition.

If function = 0, the indicated string contains the name of the program to execute with these special parameters. If function = 1, the string contains the program code itself. Just don't do anything complicated in it. Do not recurse and certainly try not to call this command inside any program or subroutines you've already called this command in. If you attempt to do so, the original calling routine will be lost.

Technical Details**Arguments**

det(25,function,Str?) ****NOT FULLY IMPLEMENTED****

Outputs

Executes BASIC code safely.

MatToStr

<http://dcs.cemetech.net/index.php?title=BasicLibs:MatToStr>

Description

Outputs a hex string based on the input matrix, for use with PicArc's STRINGTILE command. Any entry in a matrix over 255 will be treated as matrix_element modulo 256; for example, 511=1023=255.

Technical Details**Arguments**

det(26,Matrix#)

Outputs

Hex string based on input matrix.

StringRead

<http://dcs.cemetech.net/index.php?title=BasicLibs:StringRead>

Description

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

Technical Details

Arguments

det(27,"binstring",start,readthismany)

Outputs

Substring is returned.

HexToDec

<http://dcs.cemetech.net/index.php?title=BasicLibs:HexToDec>

Description

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

Technical Details

Arguments

det(28,"HEXSTRING")

Outputs

Returns integer converted from hex string.

DecToHex

<http://dcs.cemetech.net/index.php?title=BasicLibs:DecToHex>

Description

Converts a number between 0 and 65,536 to its hexadecimal equivalent. The number of hexadecimal output to the string will have its leading zeroes stripped so inputting 15 will result in "F" and 16 will result in "10". Set autooverride to 1 to output all leading zeroes, which may be useful for routines that require four hex digits at all times but cannot spend the memory/time whipping up a BASIC solution to fill the missing zeroes.

Technical Details

Arguments

det(29,SomeRealNumber,[autooverride])

Outputs

Returns up to four digits of hexadecimal in a string.

EditWord

[**Description**](http://dcs.cemetech.net/index.php?title=BasicLibs>EditWord</p></div><div data-bbox=)

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

This command, otherwise, works just like EDIT1BYTE. Its documentation is rewritten here for convenience. Replaces a word in some string variable, Str1 to Str0, with a replacement value 0 thru 65535 starting at some specified byte (start is at 0). The string supplied is edited directly so there's no output, per se. See EDIT1BYTE for more details.

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

Technical Details**Arguments**

det(30,Str?,starting_byte,replace_with_this_word)

Outputs

String is modified. However, there is no explicit return value.

BitOperate

<http://dcs.cemetech.net/index.php?title=BasicLibs:BitOperate>

Description

Performs a bitwise operation between value1 and value2 using a supplied logic value. Logic: NONE=0 AND=1 ; OR=2 ; XOR=3. These numbers are converted to a 16-bit number and then the operation is carried out in order to maintain simplicity.

If the numbers are out of bounds, then the function will exit out with an error. If a negative number is introduced, it will be treated as 65536-abs(value). This command really helps mask out hex digits but if you use strings to store those digits, you'll need to use the HEXTODEC command for each value you need. This can easily be done inline, as the example shows:

```
det ( 31 , det ( 28 , "AAAA" ) , det ( 28 , "5555" ) , 2 )
```

That example should turn out to be 65535, or "FFFF" if you were to extend the idea of functions:

```
det ( 29 , det ( 31 , det ( 28 , "AAAA" ) , det ( 28 , "5555" ) , 2 ) )
```

Technical Details

Arguments

det(31,value1,value2,logic)

Outputs

Returns a number based on bitwise math on the inputs.

GetProgList

<http://dcs.cemetech.net/index.php?title=BasicLibs:GetProgList>

Description

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

```
det ( 32 , "TEMP" )
```

would return all program names that start with the characters "TEMP", which may be something like "TEMP001 " or "TEMP001 TEMP002 TEMP003 ", etc. If SearchType = 1, appvars will be searched for; if SearchType = 2, groups will be searched for. This function will not look for lists. This is easy enough to do using other BASIC commands and Celtic III commands.

NOTE: This command is NOT to be confused with FINDPROG, which outputs a string consisting of files whose CONTENTS starts with the specified string. Also use the fact that the final name in the list is terminated with a space to make extracting names from the list easier.

Unlike all the other commands, this one will not find hidden variables. You'll have to manually use the starting character that the file was hidden by. Celtic III gives you all the tools needed to do this but you'll need to read the CIII tutorial to find out how to do this.

Technical Details

Arguments

det(32,"SERACHSTRING",SearchType)

Outputs

Get filtered list of programs on the calculator.

PicArc Library Routines

DBQuery1

<http://dcs.cemetech.net/index.php?title=BasicLibs:DBQuery1>

Description

A nifty little function that allows many kinds of operations to take place with the PicArc database. The db_argument will vary depending on what you are going to do, but not by that much. Just read below.

If function=

- 0:** Format this database. Just like anything that says "FORMAT", you lose all preexisting data if that file happens to already exist. This is not an undoable action, so take care. If the file does not already exist, it is created and then formatted. After you use this command, you will be able to work around with the file as a PicArc database. No arguments needed
- 1:** Append a picture file. db_arg1 will be the pic number you want to refer to. For Pic1, argument will be 1, for Pic9, it'll be 9. For Pic0, it'll be 10. Keep in mind that this command only appends Pic files. No insertion is available, so be careful on the order of Pic files you append.
- 2:** Delete an entry. Starting at db_arg1 = 0, this command removes the specified entry off of the database. This command is NOT undoable, so exercise extreme caution.
- 3:** Retrieves the number of entries in the database.
- 4:** Works just like function 1, except that this command will attempt to compress the pic. It automatically chooses the method of best compression so you'll always get the smallest entry that the application is capable of making, but using this command may be a little slower than its counterpart. All the other commands transparently decompresses compressed entries without much of a speed loss.
- 5:** Copies the entry denoted by db_arg1 to the graph buffer with the logic indicated by db_arg2. 0: REPLACE, no screen update. 1: AND, no screen update. 2: OR, no screen update. 3: XOR, no screen update. 4: REPLACE, the screen is updated. 5: AND, the screen is updated. 6: OR, the screen is updated. 7: XOR, the screen is updated.
- 6:** Copies an entry from the database as denoted by db_arg1 (starting at zero) to a Pic file as denoted by db_arg2. For example. to copy entry 6 of the database "FOO" to Pic1, do the following: identity(1,"FOO",6,6,1)

Technical Details

Arguments

identity(0,"DATABASENAME",function,db_arg1,db_arg2)

Outputs

Query database

TogglePic

<http://dcs.cemetech.net/index.php?title=BasicLibs:TogglePic>

Description

This command takes Pic X (Pic0 = 10) and performs an action designated by a function number. If arg_1 = ...

0: Copies Pic file to the graph buffer using logic method. arg_1 specifies the Pic file arg_2 specifies the logic code. The supported codes are:

- 0: REPLACE, no screen update.
- 1: AND, no screen update.
- 2: OR, no screen update.
- 3: XOR, no screen update.
- 4: REPLACE, the screen is updated.
- 5: AND, the screen is updated.
- 6: OR, the screen is updated.
- 7: XOR, the screen is updated.

1: Creates a new pic file using the contents of the graph buffer. Any preexisting picture will be replaced.

2: Swaps a pic file between archive and RAM. If it is now in RAM, the output is 0, else it is some other number.

3: Deletes a picture. This is not an undoable action, so be cautious while using this function.

Technical Details

Arguments

identity(1,pic_number,function,[arg_1,arg_2...])

Outputs

Modifies the LCD contents. See above

GetPicGroup

<http://dcs.cemetech.net/index.php?title=BasicLibs:ToggleSet>

Description

Outputs a list containing all the names of the pic files in the specified TI-OS group file. Each name is denoted by a number, so 1 denotes Pic1, 0 denotes Pic10, and 68 denotes Pic68.

Technical Details

Arguments

identity(2,"GROUPFILENAME")

Outputs

List of the Pics in a group file.

ExtPicGroup

<http://dcs.cemetech.net/index.php?title=BasicLibs:ExtPicGroup>

Description

Copies a pic by name from the group and puts it into a pic file of its corresponding number. Any preexisting pic files are overwritten.

Technical Details**Arguments**

identity(3,"GROUPFILENAME",pic_name_in_group)

Outputs

Copy a Pic from a group into RAM. It is not possible to copy to a Pic name other than the one specified for that Pic inside the group.

StringTile

<http://dcs.cemetech.net/index.php?title=BasicLibs:StringTile>

Description

See xLIB command DrawTileMap for information regarding the inputs. The only difference is that "Matrix_name" is replaced with "BINARYSTRING". For this, you supply a hex string converted to binary with the HexToBin command. The Height and Width property of the command is used to provide a two-dimensional matrix feel to an inherently one-dimensional structure that is a string.

If 2ByteMode is set to something other than 0, then the input string is considered words instead of bytes. This allows the user to use 4 hex digits (two bytes) per tile so up to 65536 different tiles can be accessed using this command. All other arguments function as they do with the xLIB command.

As a developer, you should develop your tilemaps in hex and then use the HexToBin command to convert it to the format required by this command. To edit this tilemap, you should use the Edit1Byte command while this tilemap is in its binary format. Note that any missing arguments will default to the value of zero (0) instead of 32 as in xLIB.

Technical Details

Arguments

identity(4,"BINSTR",xPos,yPos,Width,Height,StartX,EndX,SStartY,SEndY,Pic#,Logic,TileSize,Update_LCD,2ByteMode)

Outputs

Draws the specified tilemap to the graph buffer.

PutSprite

<http://dcs.cemetech.net/index.php?title=BasicLibs:PutSprite>

Description

This command works just like the xLIB command real(1,...) (DrawSprite) except that the Pic and the coordinates on that Pic file are not defined. Instead, a string consisting of hex digits is used to define the sprite as inline data. For example, if you wanted to draw a black 8*8 block at the top-left corner of the screen with XOR logic and drawn immediately:

```
identity(5,"FFFFFFFFFFFFFFFF",0,0,1,8,3,0,1)
```

This is useful for those that want to display sprites without the use of bulky image files. For large sprites, each byte goes LEFT first, then DOWN, so specifying "80FF0180000180000180FF01" would be rendered as a perfect box 3 bytes wide and 4 pixels tall.

Note that any missing arguments will default to the value of zero (0) instead of 32 as in xLIB.

Technical Details**Arguments**

identity(5,"HEXSTRING",x,y,w,h,logic,flip,update_lcd)

Outputs

Draws the sprite to the LCD buffer, optionally also updating the LCD.

ShiftScreen

<http://dcs.cemetech.net/index.php?title=BasicLibs:ShiftScreen>

Description

Shifts the image on the graphbuffer a number of times in a direction indicated below:

- left : 1
- right : 2
- up : 4
- up-left : 5
- up-right : 6
- down : 8

down-left : 9
down-right : 10

Note that any area that does not have new contents shifted onto it, for example the last row of the LCD if the image is shifted up, will be left with its existing contents. If screen_update is zero, the screen will not update. Otherwise, it will update.

Technical Details

Arguments

identity(6,shift_number_of_times,direction,screen_update)

Outputs

Shifts the screen in one of eight possible directions.

DrawBox

<http://dcs.cemetech.net/index.php?title=BasicLibs:DrawBox>

Description

Draws a box which corners are (x_left,Y_top) and (x_right,y_bottom) using a specified display method. Since this display method code is not complete, let's say that it is xLIB-compatible with the drawshape command, minus three. Add 128 to this number and the screen will also be updated. In addition, the code to support this command in Doors CS is a complete replacement of Celtic's code for the same, and should fix almost all the bugs with this command present in Celtic.

Technical Details

Arguments

identity(7,x_left,y_top,x_right,y_bottom,display_method)

Outputs

Draw a rectangle to the graph buffer, and optionally also the LCD

FillMap

<http://dcs.cemetech.net/index.php?title=BasicLibs:FillMap>

Description

A command that fills the screen with the same 8 x 8 tiles as defined by 16 hex digits. Along with the right and down shift arguments, this command makes scrolling backgrounds far easier. The hexstring, as said before, is 16 digits long, 8 pairs which makes up an 8 by 8 pixel tile. The right and down arguments determine how many pixels the filled map should be shifted. You can either choose not to shift or shift in either direction up to 8 pixels. Since the routine does mod 8, you can actually have these numbers increment and decrement

bounded by -65565 to 65535 so you should not actually have to test for bounds save for those already mentioned.

LOGIC applies a method used to write the tiles to the buffer. 0=overwrite ; 1=AND ; 2=OR ; 3=XOR. If UpdateLCD is anything other than zero, the screen is updated with whatever was just written to the buffer.

Technical Details

The bounds are actually -99999 and 99999 and it'll still function properly due to how the custom float to integer routine works. This "bug" is not likely to be fixed. It's not hurting anything. It's safe. No one is going to wait at a menu for as long as it takes to cycle through the routine *that* many times.

Arguments

identity(8,"HEXSTRING",Right,Down,LOGIC,UpdateLCD)

Outputs

Fills the screen with the given tiles.

BoxShift

<http://dcs.cemetech.net/index.php?title=BasicLibs:BoxShift>

Description

Note: this command is incompletely implemented

This command shifts or rotates within a box-shaped area on the screen (and may be the whole screen if you want). If you don't update the LCD, this command is very useful for dynamically editing tilemaps by loading them to the buffer, using this command to edit them, then storing them back to the pic file, all without the buffer ever being updated. The procedure is fast enough to be dismissed as a slight slowdown due to the BASIC parser.

Technical Details

This routine does not perform clipping. All coordinates and dimensions must remain onscreen.

Arguments

identity(9,x,y,w,h,Dir,Type,Repeat,LCD_update)

x: top-left corner x, 0 thru 11. (each 8 pixels is a byte)

y: top-left corner y, 0 thru 63.

w: width of box object, 1 thru 12. Specifies byte widths.

h: height of box object, 1 thru 64

Dir: Each will have a basic direction followed by what you must add to it in order to perform a different type of shift.


```
left = 1
right = 2
up = 4
up-left = 5
up-right = 6
down = 8
down-left = 9
down-right = 10
```

Type: Determines what type of shifting is to take place:

- 0 = Keep bit being shifted in
- 1 = Shift in a black bit (for black backgrounds)
- 2 = Shift in a white bit (for white backgrounds)
- 3 = Let bit that shifts out reappear on other side (for scrolling backgrounds)

Repeat: A number that repeats the shift operation as many times as specified. Cannot be zero.

LCD_update: If it is a number other than zero, the screen is updated with the results.

Outputs

Modifies the graph buffer and optionally the LCD.

DrawText

<http://dcs.cemetech.net/index.php?title=BasicLibs:DrawText>

Description

Draws text to the graph screen with the coordinates x,y using the small font. The screen is NOT updated. If you wanted to update it, you should've used the Text() command instead. If you omit the x,y coords, the text is drawn to the last coordinate the previous DRAWTEXT command ended up. Does not line-wrap. It's your duty to do that yourself. If you don't want the text to be modified in any way, set flag to zero. If you want to modify, you should do addition to the value using these numbers:

- +1** = Draw using large font.
- +2** = Erase 1 row of pixels below the small font base
- +4** = Invert text
- +8** = Hex string is used in place of text to use font map. If you know the character's hex value, you can use it. Advantage: Get chars you can't access any other way.
- +16** = Update screen anyway. Just a throwback.

Example: I want to use the large font AND invert text at x=4 and y=8 using "THIS IS TEXT" as text.

Doors CS 7 SDK

```
identity(10,0+1+4,4,8,"THIS IS TEXT")  
- or -  
identity(10,5,4,8,"THIS IS TEXT")
```

Note that flag sums have been collapsed for simplicity

Technical Details**Arguments**

```
identity(10,flag,[x,y],"TEXT")
```

Outputs

Put text into the graph buffer.

CopyBox

<http://dcs.cemetech.net/index.php?title=BasicLibs:CopyBox>

Description**Not yet implemented**

Not implemented yet. Hope to implement it soon, as it will be one of the more useful functions in the Celtic III PicArc extension code.

Technical Details**Arguments**

```
identity(11,...)
```

Outputs

Copy a section of the screen, to be completed in a future version.

XLib Compatibility Library Functions

ClearScreen

<http://dcs.cemetech.net/index.php?title=BasicLibs:ClearScreen>

Description

Clears off the graph buffer. Also used to check whether or not XLib is installed.

Technical Details

Arguments

real(0,update_lcd)

Update_lcd:

0: Do not update

1: Update the cleared screen

Outputs

See Description

DrawSprite

<http://dcs.cemetech.net/index.php?title=BasicLibs:DrawSprite>

Description

Draws a sprite to the buffer at a specified location. The sprite is taken from a Pic file. This will work even if the Pic file is archived.

Technical Details

Arguments

real(1, Spr_X, Spr_Y, Spr_Width, Spr_Height, sPIC_Num,sPIC_X, sPIC_Y, Spr_Method, Spr_Flip, Spr_UpdateLCD)

Spr_X: Upper-right of sprite to be displayed, onscreen, between 0 and 95

Spr_Y: Upper-right of sprite to be displayed, between 0 and 63

Spr_Width: Sprite width in bytes. An 8-pixel-wide sprite is 1 byte wide, 16 pixels wide is 2 bytes, and so on.

Spr_Height: Sprite height in pixels

sPIC_Num: The picture number that this sprite is stored in. 1-9 for Pic1-Pic9, 0 for Pic10.

sPIC_X: The offset for the stored sprite in the pic horizontally. Must be a value from 0-11, and is aligned on bytes (with every 8 pixels).

sPIC_Y: The offset for the sprite in the pic vertically. Does not need to be aligned.

Spr_Method:

0: overwrite

1: AND logic

2: OR logic
 3: XOR logic.
 Spr_Flip:
 0: No flip
 1: Horizontal Flip
 Spr_UpdateLCD:
 0: Do not update
 1: Update the cleared screen

Outputs

See Description

DrawTileMap
<http://dcs.cemetech.net/index.php?title=BasicLibs:DrawTileMap>
Description

Draws a tile map to the screen based off of the given matrix and picture. This routine supports up to 65535 tiles, each starting from the Pic number you specified and following to each numerically adjacent Pic # for each overflow. Let's say your starting Pic# is 5 and you want to access tile 4324. There are 96 tiles per pic on 8x8 mode, so tile 4324 will be on pic 50. Pic data will be read from archive if need be. It'll be slower, though. An additional feature added into the sprite routine is the ability to invert the sprite prior to displaying. To do this, add 4 to whatever you use for LOGIC. For example, to invert a sprite and use XOR display logic, use 7 since $3 + 4 = 7$.

Technical Details**Arguments**

real(2,Matrix_number,x_offset,y_offset,Width,Height,SStartX,SEndX,SStartY,SEndY,Pic#,Logic,TileSize,Update_LCD)

Matrix_number: Number of matrix with map data. 0=[A], 9=[J]

x_offset: x-offset for the part of map you want to draw.

y_offset: y-offset for the part of map you want to draw.

Width: width of the tilemap

Height: height of the tilemap

SStartX: column to start drawing. 0-12 (8x8) or 0-6 (16x16)

SEndX: column to end drawing. 0-12 (8x8) or 0-6 (16x16)

SStartY: row to start drawing. 0-12 (8x8) or 0-6 (16x16)

SEndY: row to end drawing. 0-12 (8x8) or 0-6 (16x16)

Pic#: Pic file to start looking for tiles from.

Logic:

0: Overwrite logic

- 1: And logic
- 2: Or logic
- 3: Xor logic

tile_size: set to 16 for 16x16 tiles. Otherwise, it's 8x8.

Update_LCD: Does not update the screen if 0, otherwise it does.

The SStartX, SEndX, SStartY, and SEndY allow you to draw a "windowed" map so you can fit things on the side, like a HUD or something. Drawing will also be accelerated for smaller maps. For example, if you wanted to draw an 8x8 tiled sized map with a 1 tile blank border on all sides:

SStartX: 1
SEndX: 11
SStartY: 1
SEndY: 7

Outputs

See Description

RecallPic

<http://dcs.cemetech.net/index.php?title=BasicLibs:RecallPic>

Description

Recalls a Pic into the graph buffer. This will work even if the Pic is archived

Technical Details

Arguments

real(3, rPIC_Num, rPIC_Method, Recall_UpdateLCD[, Full_Size])

rPIC_Num: The pic to recall. Pics from 0-255 are valid, instead of the usual Pic0-Pic9

rPIC_Method:

- 0: Overwrite
- 1: AND logic
- 2: OR logic
- 3: XOR Logic

Update_lcd:

- 0: Do not update
- 1: Update the cleared screen

Full-Size: If this is 1, then the created image is 64 rows high. If this argument is 0 or omitted, then the created image is the BASIC-standard 63 rows high.

Outputs

See Description

ScrollScreen

<http://dcs.cemetech.net/index.php?title=BasicLibs:ScrollScreen>

Description

Scrolls the screen in any of 8 directions

Technical Details

Arguments

real(4, scroll_direction, scroll_step, update_LCD)

scroll_direction:

- 0: Up
- 1: Down
- 2: Left
- 3: Right
- 4: UpLeft
- 5: UpRight
- 6: DownLeft
- 7: DownRight

scroll_step: The number pixels to scroll

Update_lcd:

- 0: Do not update
- 1: Update the cleared screen

Outputs

See Description

ChngContrast

<http://dcs.cemetech.net/index.php?title=BasicLibs:ChngContrast>

Description

Sets or gets the current contrast

Technical Details

Arguments

real(5, function, value)

function:

- 0: Set contrast
- 1: Current contrast is returned by the function into Ans, (value argument is ignored for this).

value: A value 0-39, where 0 is the lightest, and 39 is the darkest.

Outputs

Sets or gets the contrast, depending on **function**.

UpdateLCD

<http://dcs.cemetech.net/index.php?title=BasicLibs:UpdateLCD>

Description

Updates the screen with any changes made since the last update.

Technical Details**Arguments**

real(6)

Outputs

LCD updated.

RunIndicator

<http://dcs.cemetech.net/index.php?title=BasicLibs:RunIndicator>

Description

Turns on or off the run indicator.

Technical Details**Arguments**

real(7, value)

value:

0: Off

1: On

Outputs

Run indicator enabled or disabled.

GetKey

<http://dcs.cemetech.net/index.php?title=BasicLibs:GetKey>

Description

Reads the keypad for any keypress. This function will not block, and it'll return zero if no keys were pressed at the time. This routine supports multiple arrow key presses.

The key codes are as follows as it appears on the keyboard:

F1	F2	F3	F4	F5
[53]	----	[52]	----	[51]
2nd	MODE	DEL	<div> <div>/ 7 Up 8 \</div> <div>/ 4 \</div> <div>Left 2 3 Right</div> <div>\ 1 /</div> <div>5 Down 6</div> </div>	
[54]	----	[55]	----	[56]
Alpha	x,T,0,n	STAT		
[48]	----	[40]	----	[32]
MATH	APPS	PRGM	VARS	CLEAR
[47]	----	[39]	----	[15]
X ⁻¹	SIN	COS	TAN	^
[46]	----	[38]	----	[14]
X ²	,	()	/
[45]	----	[37]	----	[13]
LOG	7	8	9	*
[44]	----	[36]	----	[12]
LN	4	5	6	-
[43]	----	[35]	----	[11]
STO	1	2	3	+
[42]	----	[34]	----	[10]
ON	0	.	(-)	ENTER
[N/A]	----	[33]	----	[9]

Other (unofficial) codes:

57 = left/right

58 = left/right/down

59 = up/down

60 = up/left/down

61 = up/right/down

62 = up/right/left

63 = all arrow keys mashed

Technical Details

Arguments

real(8)

Outputs

Numeric keycode, as above.

CreatePic

<http://dcs.cemetech.net/index.php?title=BasicLibs:CreatePic>

Description

Draws or deletes a picture variable.

Technical Details

Arguments

`real(9, function, pic#[,full-size])`

function:

- 0: Update the screen from buffer and copy what's there
- 1: Delete Pic (not undoable. Exercise caution)
- 2: Copy image from buffer to pic without updating screen

Pic#: a number between 1 and 255, corresponding to the pic number. pic0=10

Full-Size: If this is 1, then the created image is 64 rows high. If this argument is 0 or omitted, then the created image is the BASIC-standard 63 rows high.

Outputs

See Description

ExecArcPrgm

<http://dcs.cemetech.net/index.php?title=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":real(10,0,2):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":real(10,0,12):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:

```
real(10,1,2)
```

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:

```
real(10,2)
```

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

Technical Details

Arguments

```
"PRGMNAME
```

```
real(10,function,temp_prog_number)
```

```
prgmXTEMP0XX or :Asm(prgmXTEMP0XX), depending on format.
```

Outputs

See Description

GetCalcVer

<http://dcs.cemetech.net/index.php?title=BasicLibs:GetCalcVer>

Description

Outputs a number to Ans indicating which type of calculator is being used:

0: 83+

1: 83+ SE

2: 84+

3: 84+ SE

Technical Details

Arguments

```
real(11)
```

Outputs

Number indicating calculator model

DrawShape

<http://dcs.cemetech.net/index.php?title=BasicLibs:DrawShape>

Description

Draws a shape define by points (x1,y1) and (x2,y2). If update_LCD is 0, the screen will not be updated, otherwise it will.

Technical Details

Arguments

```
real(12,shape_type,x1,y1,x2,y2,update_LCD)
```

shape_type:

- 0 = Draw a single BLACK LINE.
- 1 = Draw a single WHITE LINE.
- 2 = Draw a single INVERTED LINE.
- 3 = Draw an EMPTY RECTANGLE with BLACK OUTLINE
- 4 = Draw an EMPTY RECTANGLE with WHITE OUTLINE
- 5 = Draw an EMPTY RECTANGLE with INVERTED OUTLINE
- 6 = Draw a RECTANGLE with a BLACK FILLING
- 7 = Draw a RECTANGLE with a WHITE FILLING
- 8 = Draw a RECTANGLE with an INVERTED FILLING
- 9 = Draw a WHITE RECTANGLE with a BLACK OUTLINE
- 10 = Draw a BLACK RECTANGLE with a WHITE OUTLINE

x1: top left x coordinate

y1: top left y coordinate

x2: bottom x coordinate

y2: bottom y coordinate

Update_lcd:

- 0: Do not update
- 1: Update the cleared screen

Outputs

Shape drawn to graph buffer and possibly LCD, if specified.

TextMode

<http://dcs.cemetech.net/index.php?title=BasicLibs:TextMode>

Description

Performs a text-related function, if function is:

- 0: Unsets the text inverse function
- 1: Sets text inverse
- 2: Sets lowercase access. (press Alpha twice)
- 3: Unsets lowercase support
- 4: See note below.

Some menus in the TI-OS will reset these flags.

Note: For command 4, you can output a character onto the screen given a character ASCII code, some number between 0 and 255. As a hint, the space is 32, the upper-case letter "A" is 65, and the checkerboard cursor is 241.

Technical Details**Arguments**

real(13,function,[character])

Outputs

See description

CheckRAM

<http://dcs.cemetech.net/index.php?title=BasicLibs:CheckRAM>

Description

Returns the amount of free RAM, measured in bytes.

Technical Details

Arguments

real(14)

Outputs

Bytes of free RAM.

Omnicalc Compatibility Routines

Sprite

<http://dcs.cemetech.net/index.php?title=BasicLibs:Sprite>

Description

Emulates the Omnicalc function sprite().

Technical Details

Arguments

real(20,pic#,picX,picY,w,h,x,y,[logic])

Outputs

Sprite drawn.

ExecAsm

<http://dcs.cemetech.net/index.php?title=BasicLibs:ExecAsm>

Description

Executes the given string of hex. Does **not** automatically append \$C9.

Technical Details

Arguments

real(33,"HEXSTRING")

Outputs

Hex executed

CHAPTER 4

OVERVIEW OF DOORS CS ASM TOOLS AND FEATURES

Doors CS 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.cemetechnet.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 7 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 CS, bugfixes are occasionally implemented, and mistakes are sometimes found. Be sure to frequent the Doors CS wiki at <http://dcs.cemetechnet.net> to view the latest documentation and look for new versions of Doors CS.

Assembling Suite and Compile.bat/Compile.sh

In the interest of making the process of creating programs for Doors CS 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, and if it succeeds, create a binary object file with my BinPac8x v1.3 tool. 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
```

You **must** have Python 2.7 or higher or Python 3.1 or higher on your computer in order to use the BinPac8x object creator.

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
```

In some circumstances it may be necessary to `chmod +x compile.sh`. You **must** have the program “mono” to assemble using Brass, and Python 2.7 or higher or Python 3.1 or higher in order to use the BinPac8x object creator. Please consult your distribution’s documentation about how to install packages.

Doors CS 7 Include File

The Doors CS 7 include file (dcs7.inc) is optimized for use with the included compilation package. Simply `#include “dcs7.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 file defines all of the routines in this document for ASM programmers, as well as constants for general use and for GUI API users.

Doors CS Tools for ASM Programmers

Types of Programs

There are four main types of “programs” that programmers can write for Doors CS. The most common type is a simple executable program, with a header optionally defining an icon, a description, and shared libraries to be used. A more complex type of executable program can open files of a specified type, just like Notepad in Windows can open .txt files but [browser of your choice] opens .html files. Finally, there are Shell Expansions (SEs), small to medium-sized modules that plug into Doors CS and provide realtime functions while in Doors CS itself, and Appended Library Extensions (ALEs), which are used to share routines and code among several programs.

Program Headers

The Doors CS header for simple executables follows a fairly standard form. There are some variations based on whether you want to include an icon or a description, or use any Appended Library Extensions (ALEs), the latter of which are all but deprecated at this point. Programs utilizing the Associated Program system that lets Doors CS automatically open files with an associated viewer take an expanded header, as described in the [chapter](#) on the subject. Without further ado, the most minimal Doors CS header:

```
; Program Name:
; Author:
; Version:
; Date:
; Written for Doors CS 7.0 and higher (http://dcs.cemetech.net)

.nolist
#include "ti83plus.inc"
#include "dcs7.inc"
.list

.org progstart          ;.equ $9d93, by the way
.db $BB,$6D

Init:
    xor d
    ret
    jr Start

    .dw $0000            ;No Description
    .db $07,$00          ;always this string
    .dw $0000            ;No Icon
    .dw $0000            ;No ALEs
Start:                  ;main routines
    ;etc etc etc
```

This program takes advantage of almost no features of the Doors CS header, other than specify that it is a DCS 7.0 program. The opposite end of the spectrum contains an icon, a description, and ALEs, as shown below. Most programs written should have an icon and a description, but will not use ALEs.


```

; Program Name:
; Author:
; Version:
; Date:
; Written for Doors CS 7.0 and higher (http://dcs.cemetech.net)

.nolist
#include "ti83plus.inc"
#include "dcs7.inc"
.list

.org progstart          ;.equ $9d93, by the way
.db $BB,$6D

Init:
    xor d
    ret
    jr Start

    .dw Description      ;or .dw $0000 if you don't have a desc
    .db $07,$00          ;always this string
    .dw Icon             ;or .dw $0000 if you don't have an icon
    .dw ALE              ;usually .dw $0000 if you don't
                        ;have or know what an ALE is
Start:
                        ;main routines
    ;etc etc etc
    ;ret
;The following is at the end of the program
Description:
    .db "Description",0  ;can be omitted if .dw Description
                        ;is .dw 0000 above
Icon:
                        ;a 16x16 icon (can be omitted if
                        ;.dw Icon is .dw 0000 above)
    .db %11111111,%11111000
    .db %10000000,%00001100
    .db %10111100,%00001010
    .db %10010010,%00001111
    .db %10010010,%01010001
    .db %10010010,%10101001
    .db %10111100,%01010101
    .db %10000000,%00000001
    .db %10101010,%10101001
    .db %10010101,%01010101
    .db %10101010,%10101001
    .db %10000000,%00000001
    .db %10010101,%01010101
    .db %10101010,%10101001
    .db %10000000,%00000001
    .db %11111111,%11111111
ALE:
                        ;must be omitted if .dw ALE is .dw
0000 above
    .db "ZALE",0,0,0,0  ;always eight bytes, use
    .db "ZLALE",0,0,0  ;zeros for extra bytes

```

The GUI API

The Associated Program System

As mentioned briefly above, the Associated Program system lets your program claim one or more filetypes as files it knows how to open (and can potentially save). There is a special header format to deal with this case, which allows Doors CS to do all the dirty work with copying data to RAM, providing pointers, cleaning up, and performing writeback when necessary. There is a whole chapter in this document devoted to explaining the Associated Program system.

Other Features

Doors CS provides many other powerful systems, such as a RunProg routine that can execute any program recursively, be it ASM or BASIC, an executable or an AP file, then eventually return control to your program when the called program completes. It contained now-defunct networking drivers that will be resurrected in the near future. It clones the MirageOS tasker interrupt that can perform supervisory tasks for programs.

Doors CS contains a wealth of features, many of which have existed for so long or are relatively minor and have been lost in the shuffle of development. Check the Doors CS wiki for any updated information omitted from or abbreviated in this document.

CHAPTER 5

THE DOORS CS GUI API

The Doors CS GUI API uses an innovative stack-based system to handle the creation, rendering, and use of GUI-enabled programs. This system has a twofold advantage. Firstly, the sequential order makes it easy to "push" new windows or elements to the top above existing GUI elements, then "pop" them off to return to a backgrounded window. Secondly, by identifying each chunk of data on the stack with a size word and type byte, the stack can easily be traversed backwards and forwards for parsing and properly handle dynamic sizing of array elements.

In utilizing the stack, you will be addressing elements by index from the bottom of the stack, not by absolute or relative memory location, thus avoiding nearly all the problems of dynamic memory allocation. Without further ado, let's go through the steps of creating, rendering, and handling a GUI element. As with all other sections of this SDK, you can find the most up-to-date version of this chapter at http://dcs.cemetech.net/index.php?title=GUI_API.

Working with the GUI API

Initializing the GUI

Before you can do anything with the GUI, you must initialize it by calling `OpenGUIStack`. This will create an Appvar called `gui6` (if it doesn't already exist) and initialize its contents to an empty stack. If in doubt whether the GUI Stack is open yet, call this function anyway; it will silently handle a stack already open and report this state in register `c`. Once the stack is open, you can begin to push elements onto it.

Adding GUI Elements

The function for adding elements to the top of the stack is `PushGUIStack`, defined in `dcs7.inc`. It takes three arguments, a pointer to the data to be added in `hl`, the size of the data to add in `de`, and the type in the accumulator, `a`. All of the different types are defined in `dcs7.inc` and can also be referred to numerically as outlined in the documentation for each GUI type. The data pointed to by `hl` should only contain the data to be added, not the three-byte header that Doors CS adds to the GUI Stack before the data. For example, suppose you wanted to add a small window to the current GUI:

```
ld hl,SmWinData
ld de,SmWinDataEnd-SmWinData
ld a,GUIRSmallWin
call PushGUIStack
...
SmWinData:
    .db 5,5      ;the x and y coordinates relative to the LCD of the
                ;top-left of the window
    .db $F8,$88,$88,$88,$F8 ;a square icon
    .db "My Window",0      ;the window title
SmWinDataEnd:
```

Space-Efficiency: Adding Multiple Elements at Once

If you want to add multiple elements to the GUI at once without repeatedly calling the PushGUIStack function, an alternative function called PushGUIStacks exists (note the pluralization). This takes only one argument, a pointer in hl to a specially-formatted mass of GUI data. The data must contain one or more GUI elements, each specified as a 2-byte size word, a one-byte type byte, and then the one-or-more byte GUI element. The size word should be three bytes larger than if it was the value of de being passed to PushGUIStack; in other words, it represents the size of this item including the type byte and the size word itself. Additional elements in the sequence to be pushed follow the same format. After the last item, to signal that the end of the sequence has been reached, two bytes of \$ff (.db \$ff,\$ff) should be used. The following two-item sequence example is taken from within Doors CS itself:

```
SimpleGUIMouseData:
    .dw 4
    .db 0
    .db $fe
SimpleGUIMouseDataHS:
    .dw 6+3
    .db $0e
    .db 0,0,96,64
    .dw dbfSimpleGUIMouseClick
SimpleGUIMouseDataEnd:
    .db $ff,$ff
```

Rendering the GUI

You can choose two different methods of handling the GUI: rendering the current Stack and then handling input yourself, or handing full control over to Doors CS. If you choose the latter, Doors CS will render the Stack, initiate the mouse subsystem, then wait for the user to click on an actionable item. The function to merely render the GUI is RenderGUI, and the function to hand over full control is GUIMouse. Keep in mind that GUIMouse has a few important restrictions on its usage to maintain the stability of the users' calculators:

1. Always, **always** have a `ld hl,0` immediately before call `GUIMouse` unless you specifically are adding a mouse hook.
2. At functions jumped to as a result of clicking in `GUIMouse`, it is vital to have a call `ResetAppPage` as the **first** instruction after the label so that Doors CS is reset to the proper page.
3. Always use the following format to call `GUIMouse`. Do **not** jump to it, ever. This will fatally disturb the normal stack:

```
ld hl,0          ;this can be nonzero if you have a MouseHook
call GUIMouse
ret
```

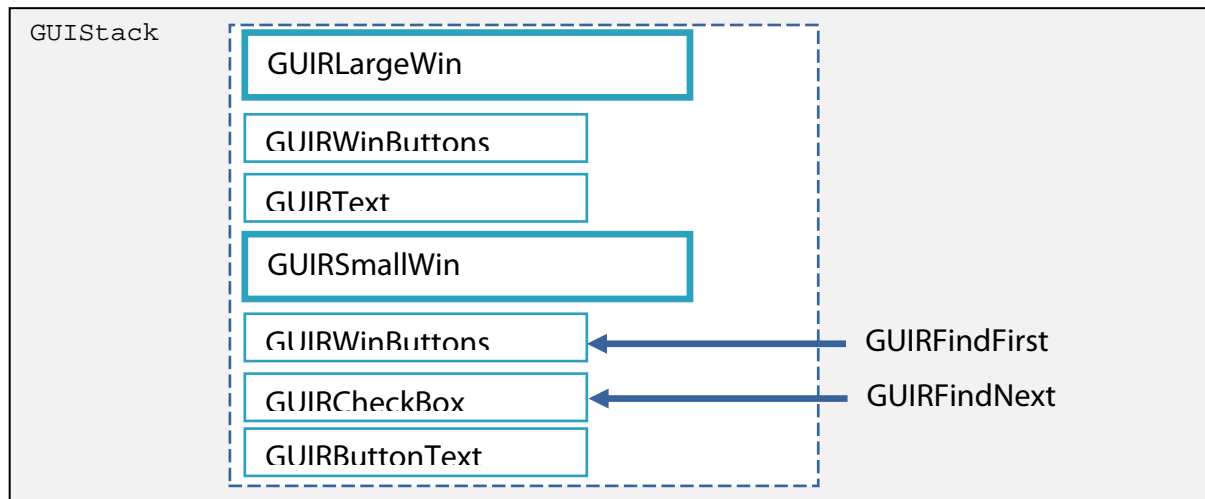
Note of course that so such caveats apply to `RenderGUI`, which simply renders the GUI and returns to the next line in your program.

Accessing Modified Data

After calling `GUIMouse`, items in the GUI Stack may have been modified, including the states of radio buttons and checkboxes and the contents of text and numerical entry fields. In order to figure out what the current data of an element in the GUI Stack is, you can use `GUIFindFirst` and `GUIFindNext`. You must always start by calling `GUIFindFirst`; this will give the address and size of the **first non-groupmaster** element on the stack. A groupmaster element is a `GUIRnull`, a `GUIRsmallWin`, or a `GUIRLargeWin`. You can then use `GUIFindNext` to find subsequent entries in the GUI Stack. Be aware that while `GUIFindFirst` skips groupmaster elements, `GUIFindNext` sequentially finds all following entries, including groupmasters.

As of Doors CS 6.6 beta or thereabouts, a third function is available called `GUIFindThis`, which works very very quickly compared with a series of `GUIFindFirst`/`GUIFindThis`/`GUIFindThis` calls, particularly if the user has a lot of programs.

The following diagram should clarify how accessing the data works. The scenario pictured involves a `GUIRLargeWin` pushed with several items, then a `GUIRsmallWin` pushed on top of it with several items of its own. The programmer calls `GUIFindFirst`, then `GUIFindThis` so that he or she can check if the checkbox is checked or not.



Removing GUI Elements

After finishing with parts of the GUI, you can remove items individually or in bulk. For a single GUI entry, use `PopGUIStack`. If you want to remove multiple elements at once, load the number of elements to be removed into register `b` and call `PopGUIStacks`. Doors CS will handle all updating of pointers, memory, and size. However, keep in mind that pointers to variables created after the GUI Stack was opened may need to be updated to take the new size of the `gui6` `Appvar` into account.

Closing the Stack

When you are done using the GUI, you can remove all GUI elements and close down the API system with the call `CloseGUIStack`. It is not necessary to pop elements off of the stack before closing it; the `CloseGUIStack` call will take care of that.

Example of a Simple GUI Program

Using the GUI system is mainly a matter of pushing a bunch of elements onto the stack, then doing things based on what is done. For example, the following code (should, it's untested) spawns a `GUIRSmallWin` at the center of the screen with a 'Crash' button which (surprise, surprise) crashes the calculator, as well as a close button in the form of `GUIRWinButtons`. This program could be made more efficient through use of the `PushGUIStacks` function, but has been left simpler and less efficient for the sake of clarity.

```

call OpenGUIStack
ld hl,myWin
ld de,crashButton-myWin
ld a,GUIRSmallWin
call PushGUIStack
;this spawns our small window, which holds all other objects
;we spawn, until we spawn another groupmaster element
ld hl,crashButton
ld de,winButtons-crashButton
ld a,GUIRButtonText
call PushGUIStack
;just spawned the 'Crash button'
ld hl,winButtons
ld de,dat_end-winButtons
ld a,GUIRWinButtons
call pushGUIStack
;added the close button to the window's title bar
ld hl,0 ;very important
call GUIMouse ;handles everything, woo
ret ;exit the program
exitMyProg:
call ResetAppPage
ret
;it jumps here upon clicking the exit button, and this returns us back
;to where the call to GUIMouse is
myWin:
.db 15 ;x coordinate of the window, 15 makes it centered
.db 15 ;same, but y coordinate
.db 0,0,0,0,0 ;just a blank icon, since I'm lazy
.db "Crash me!",0 ;the window title
crashButton:
.db 1 ;x coordinate- this is the coordinate WITHIN THE
;CURRENT GROUPEMASTER, not overall
.db 1 ;y coordinate- see above
.dw 0 ;I'm just having it jump to 00h when you click this,
;it's just the easiest way to crash it
.db "Crash!",0 ;the text to display on the button
winButtons:
.db 00100000b ;only displaying a close button
.dw 0 ;null pointer
.dw 0 ;another null pointer
.dw exitMyProg ;we'll jump to exitMyProg when this button is
clicked
dat_end:

```

GUI Memory Areas

Due to the complexity of the GUI system, several pieces of RAM may be overwritten while it is running. They are:

- 73 bytes starting at AppBackupScreen (used for GUI rendering and mouse) (for future compatibility, it is recommended to leave the first 129 bytes of AppBackupScreen free, if possible)
- 4 bytes starting at iMathPtrs (used only in the GUI mouse)
- 4 bytes starting at iMathPtrs+5 (used only in FileOpen and FileSaveAs)
- RelocatablePtr and RelocatablePtr2 (used only in FileOpen, FileSave, and FileSaveAs)

In addition, you should be aware that Doors CS will change the size and contents of the 'gui6'/'gui7' appvar during GUI execution. It will also change the contents, but not size of the DCS6/DCS7 appvar during GUI execution. It is not recommended that you use undeclared areas of the free user ram for storage while the GUI stack is open, as data corruption may occur. Declare any RAM you need as programs or appvars. Also remember to update pointers if your data file is in higher RAM than the gui6/gui7 appvar. In general, if you use the Associated Program (AP) system, Doors CS will handle keeping gui7 in higher memory than any AP file that is opened. Another important note is that the three GUI text input types, GUIRTextLineIn, GUIRPassIn, and GUIRTextMultiline all use an edit buffer, which temporarily consumes all of free RAM (but only when the user is actually typing in that text box). Therefore, be sure not to store any data in free RAM above in-RAM programs, which is already a bad idea to begin with. The most up-to-date information from this section can be found at http://dcs.cemetech.net/index.php?title=GUI_Memory_Areas.

GUI Stack Items

- [GUIRNull](#) - Provides a non-rendered invisible GUI wrapper
- [GUIRLargeWin](#) - Provides a fullscreen GUI wrapper window
- [GUIRSmallWin](#) - provides a windowed GUI wrapper
- [GUIRFullScreenImg](#) - Renders a 96x64 image to the screen
- [GUIRText](#) - Renders a single line of text
- [GUIRWinButtons](#) - Adds clickable Minimize/Maximize/Close buttons to a window
- [GUIRWrappedText](#) - Renders a wrapped chunk of text
- [GUIRButtonText](#) - Renders and handles a clickable button with text
- [GUIRButtonImg](#) - Renders and handles a clickable button with an icon
- [GUIRTextLineIn](#) - Renders and handles a plaintext single-line input element
- [GUIRRadio](#) - Renders and handles a radio button member of a group
- [GUIRCheckbox](#) - Renders and handles a checkbox
- [GUIRByteInt](#) - Renders and handles a 1-byte numerical input spinner
- [GUIRWordInt](#) - Renders and handles a 2-byte numerical input spinner
- [GUIRHotspot](#) - Creates an onClick area to an external routine
- [GUIRTextMultiline](#) - Renders and handles multiline plaintext input
- [GUIRSprite](#) - Draws a small (8 by n) sprite to the screen

- [GUIRLargeSprite](#) - Draws a large (8*m by n) sprite to the screen
- [GUIRPassIn](#) - Renders and handles a password single-line input element
- [GUIRScrollVert](#) - Renders and handles a vertical scrollbar
- [GUIRScrollHoriz](#) - Renders and handles a horizontal scrollbar
- [GUIRBorder](#) - Draws an unfilled rectangular border
- [GUIRRect](#) - Draws a filled rectangle
- [GUIRCustom](#) - Calls out to a custom rendering routine
- [GUIRMouseCursor](#) - Allows the cursor to change within a specified rectangular area

GUIRNull

Null Window Entry (Type ID 0)

<http://dcs.cemetech.net/index.php?title=GUIRnull>

Description

This item is one of the three possible items that start a GUI stack entry: GUIR Null, GUIR Small Window, and GUIR Large Window. This item only acts as a placeholder and container for the data within it; it does not render anything to the screen.

Format

Datalength: 1 byte

0: \$FF (for an opaque background) or \$FE (for a transparent background) [1 byte]

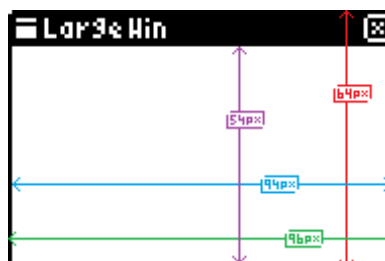
GUIRLargeWin

Large Window Item (Type ID 1)

<http://dcs.cemetech.net/index.php?title=GUIRLargeWin>

Description

This item is one of the three possible items that start a GUI stack entry: GUIR Null, GUIR Small Window, and GUIR Large Window. This item acts as a container for the data within it; it also renders a large window onto the screen.



Format

Datalength: 7+ bytes

0: 5x5 icon [5 bytes]

5: zero-terminated title string [2+ bytes]

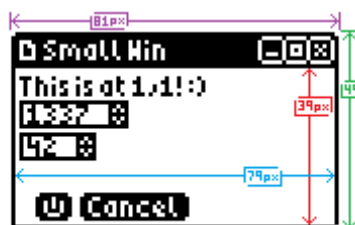
GUIRSmallWin

Small Window Item (Type ID 2)

<http://dcs.cemetech.net/index.php?title=GUIRSmallWin>

Description

This item is one of the three possible items that start a GUI stack entry: GUIR Null, GUIR Small Window, and GUIR Large Window. This item acts as a container for the data within it; it also renders a small window onto the screen, preserving any items behind it.



Format

Datalength: 9+ bytes

0: x-coordinate of upper-left (0-15) [1 byte]

1: y-coordinate of upper-left (0-15) [1 byte]

2: 5x5 icon [5 bytes]

7: zero-terminated title string [2+ bytes]

Notes

As of Doors CS 6.6 beta and above, pressing the [Clear] key on the calculator's keypad is now a shortcut to click the [X] at the top-right of a GUIRSmallWin, if such a button exists. If you wish to override this behavior, you may create a MouseHook (see GUIMouse) to catch and handle the [Clear] key.

GUIRFullScreenImg

Full-Screen Image Entry (Type ID 3)

<http://dcs.cemetech.net/index.php?title=GUIRFullScreenImg>

Description

This item should normally be used with a GUIRNull item to provide a custom background to a GUI or window. It is useful for showing non-GUI content in the background of things like a small window.

Format

Datalength: 768 bytes

0: (a 96x64 bit-encoded monochrome image) [768 bytes]

GUIRText**Static Non-Wrapping Text Entry (Type ID 4)**

<http://dcs.cemetech.net/index.php?title=GUIRText>

Description

This item places static text in the GUI using the custom Doors CS text routines. It supports multiple fonts, but this feature is not yet activated; only the default font, fontset '0', is available.

Format

Datalength: 5+ byte

0: x-coord of leftmost text column [1 byte]

1: y-coord of row above text [1 byte]

2: font (always 0 for DCS7) [1 byte]

3: zero-terminated text string [two or more bytes]

GUIRWinButtons**Clickable Window Titlebar Button Entry (Type ID 5)**

<http://dcs.cemetech.net/index.php?title=GUIRWinButtons>

Description

This item places one or more of the possible window titlebar buttons, close, maximize, and/or minimize, in the upper-right of the current GUI. It should only be used with GUIRSmallWin or GUIRLargeWin items; use with GUIRnull will probably cause a crash.

Format

Datalength: 7 bytes

0: button flags: %[-],[o],[x],00000. Any bit set means that button should be shown. [1 byte]

1: Onclick pointer for [-] (or 0 if not present) [2 bytes]

3: Onclick pointer for [o] (or 0 if not present) [2 bytes]

5: Onclick pointer for [x] (or 0 if not present) [2 bytes]

GUIRWrappedText**Static Wrapping Text Item (Type ID 6)**

<http://dcs.cemetech.net/index.php?title=GUIRWrappedText>

Description

This item places text on the screen; it will wrap at a \$D6 or if the width exceeds (width). Does not check for overflowing the bottom of the window. As of Doors CS 6.9 beta, you can add the two-byte sequence \$D6,\$01 in the middle of a string as a line break.

Format

Datalength: 6+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: width (pixels) [1 byte]

3: font (always 0 in this version) [1 byte]

4: zero-terminated text string [2+ bytes]

GUIButtonText**Clickable Text Button (Type ID 7)**

<http://dcs.cemetech.net/index.php?title=GUIButtonText>

Description

This item places a text button on the screen that jumps to a specified location in memory when it is clicked.

Format

Datalength: 6+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: Onclick pointer [2 bytes]

4: zero-terminated text string [2+ bytes]

GUIButtonImg**Clickable Image Button (Type ID 8)**

<http://dcs.cemetech.net/index.php?title=GUIButtonImg>

Description

This item places an image button on the screen that jumps to a specified location in memory when it is clicked.

Format

Datalength: 11+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: Onclick pointer [2 bytes]

4: Image width in bytes [1 byte]

5: Button width in pixels [1 byte] (account for buttons that are not x8 wide)

6: Image data [image width in bytes times 5]

GUITextLineIn**Single-Line Plaintext Interactive Input (Type ID 9)**

<http://dcs.cemetech.net/index.php?title=GUITextLineIn>

Description

This item places a single-line text input item onscreen. In the GUIMouse section, Doors CS handles all text-editing functionality.

Format

Datalength: 7+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: width in pixels [1 byte]

3: Maximum characters allows [2 bytes]

5: Current offset from start of data string [2 bytes]

7: Zero-terminated string [1+ bytes]

GUIRRadio**Grouped Radio Button (Type ID 10/\$0A)**

<http://dcs.cemetech.net/index.php?title=GUIRRadio>

Description

This item renders a radio button to the screen in a specified group and handles all clicking necessary.

Format

Datalength: 7+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: group [1 byte] (enumeration is arbitrarily defined by the programmer)

3: Current state [1 byte] (1 = selected)

4: Zero-terminated display string [1+ bytes]

GUIRCheckbox**Ungrouped Checkbox (Type ID 11/\$0B)**

<http://dcs.cemetech.net/index.php?title=GUIRCheckbox>

Description

This item renders a checkbox to the screen in a specified group and handles all clicking necessary.

Format

Datalength: 7+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: group [1 byte] (enumeration is arbitrarily defined by the programmer)

- 3: Current state [1 byte] (1 = selected)
 - 4: Zero-terminated display string [1+ bytes]
-

GUIByteInt

Byte Input Spinner (Type ID 12/\$0C)

<http://dcs.cemetech.net/index.php?title=GUIByteInt>

Description

This item allows the user to choose a numerical value no smaller than 0 nor larger than 255 within predefined limits.

Format

Datalength: 5 bytes

- 0: x-coordinate [1 byte]
 - 1: y-coordinate [1 byte]
 - 2: initial value [1 byte]
 - 3: Minimum allowed value [1 byte]
 - 4: Maximum allowed value [1 byte]
-

GUIWordInt

Word Input Spinner (Type ID 13/\$0D)

<http://dcs.cemetech.net/index.php?title=GUIWordInt>

Description

This item allows the user to choose a numerical value no smaller than 0 nor larger than 65535 within predefined limits.

Format

Datalength: 8 bytes

- 0: x-coordinate [1 byte]
 - 1: y-coordinate [1 byte]
 - 2: initial value [2 bytes]
 - 3: Minimum allowed value [2 bytes]
 - 4: Maximum allowed value [2 bytes]
-

GUIHotspot

Clickable Hotspot (Type ID 14/\$0E)

<http://dcs.cemetech.net/index.php?title=GUIHotspot>

Description

This item creates an invisible, clickable hotspot for the mouse routine.

Format

Datalength: 6 bytes

0: x-coordinate [1 byte]
1: y-coordinate [1 byte]
2: width in pixels [1 byte]
3: height in pixels [1 byte]
4: Onclick pointer [2 bytes]

GUITextMultiline

Multi-line Interactive Text Input Element (Type ID 15/\$0F)

<http://dcs.cemetech.net/index.php?title=GUITextMultiline>

Description

This item creates a text input element allowing CRLFs and more organized formatting (plus a higher character limit) than the Single-line input element.

Format

Datalength: 7+ bytes
0: x-coordinate [1 byte]
1: y-coordinate [1 byte]
2: number of rows [1 byte]
3: width in pixels [1 byte]
4: offset to start of onscreen text from start of string [2 bytes]
6: Zero-terminated string data [1+ bytes]

GUISprite

Small (8 x n) Sprite (Type ID 16/\$10)

<http://dcs.cemetech.net/index.php?title=GUISprite>

Description

This item renders an 8 pixel by n pixel sprite.

Format

Datalength: 4+ bytes
0: x-coordinate [1 byte]
1: y-coordinate [1 byte]
2: height in pixels [1 byte]
3: sprite data [height bytes]

GUIRLargeSprite

Large (8*m x n) Sprite (Type ID 17/\$11)

<http://dcs.cemetech.net/index.php?title=GUIRLargeSprite>

Description

This item renders an 8*m pixel by n pixel sprite.

Format

Datalength: 5+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: width in bytes [1 byte]

3: height in pixels [1 byte]

4: sprite data [height*width bytes]

GUIRPassIn**Single-Line Interactive Password Input (Type ID 18/\$12)**

<http://dcs.cemetech.net/index.php?title=GUIRPassIn>

Description

This item places a single-line password input item onscreen. In the GUIMouse section, Doors CS handles all text-editing functionality. Identical to Single-Line Text Input except for obscured text rendering.

Format

Datalength: 7+ bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: width in pixels [1 byte]

3: Maximum characters allows [2 bytes]

5: Current offset from start of data string [2 bytes]

6: Zero-terminated string [1+ bytes]

GUIRScrollVert**Vertical Scrollbar (Type ID 19/\$13)**

<http://dcs.cemetech.net/index.php?title=GUIRScrollVert>

Description

This item places a clickable vertical scrollbar on the screen and handles all movement clicking with associated handlers.

Note for onClick: (bc) and (bc+1) are the little-endian current values of the scrollbar position for OnScrollUp and OnScrollDown.

Format

Datalength: 16 bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: height in pixels [1 byte]

3: ID number [1 byte]

- 4: Increment/decrement per click [2 bytes]
 - 6: Minimum value on scrollbar [2 bytes]
 - 8: Maximum value on scrollbar [2 bytes]
 - 10: Current value on scrollbar [2 bytes]
 - 12: Onclick pointer for upward scrolling [2 bytes]
 - 14: Onclick pointer for downward scrolling [2 bytes]
-

GUIScrollHoriz

Horizontal Scrollbar (Type ID 20/\$14)

<http://dcs.cemetech.net/index.php?title=GUIScrollHoriz>

Description

This item places a clickable horizontal scrollbar on the screen and handles all movement clicking with associated handlers.

Note for onClick: (bc) and (bc+1) are the little-endian current values of the scrollbar position for OnScrollLeft and OnScrollRight.

Format

- Datalength: 16 bytes
- 0: x-coordinate [1 byte]
 - 1: y-coordinate [1 byte]
 - 2: height in pixels [1 byte]
 - 3: ID number [1 byte]
 - 4: Increment/decrement per click [2 bytes]
 - 6: Minimum value on scrollbar [2 bytes]
 - 8: Maximum value on scrollbar [2 bytes]
 - 10: Current value on scrollbar [2 bytes]
 - 12: Onclick pointer for leftward scrolling [2 bytes]
 - 14: Onclick pointer for rightward scrolling [2 bytes]
-

GUIRBorder

Rectangular Unfilled Border (Type ID 21/\$15)

<http://dcs.cemetech.net/index.php?title=GUIRBorder>

Description

This item renders an unfilled rectangular border onscreen.

Format

- Datalength: 5 bytes
- 0: x-coordinate [1 byte]
 - 1: y-coordinate [1 byte]
 - 2: width in pixels [1 byte]

3: width in pixels [1 byte]

4: color (0x00 or 0xFF - intermediate values created dashed lines) [1 byte]

GUIRect

Filled Rectangle (Type ID 22/\$16)

<http://dcs.cemetech.net/index.php?title=GUIRect>

Description

This item renders a filled rectangle onscreen

Format

Datalength: 5 bytes

0: x-coordinate [1 byte]

1: y-coordinate [1 byte]

2: width in pixels [1 byte]

3: width in pixels [1 byte]

4: fill color (0x00 or 0xFF - intermediate values created dashed lines) [1 byte]

GUICustom

Custom Rendering Routine (Type ID 23/\$17)

<http://dcs.cemetech.net/index.php?title=GUICustom>

Description

This item calls a custom rendering routine in the render loop.

Format

Datalength: 2 bytes

0: pointer to render routines [2 bytes]

GUIMouseCursor

User-Defined Cursor Shuffler (Type ID 24/\$18)

<http://dcs.cemetech.net/index.php?title=GUIMouseCursor>

Description

This item calls a custom rendering routine in the render loop.

Format

Datalength: 20 bytes

0: word-packed top-left y,x of affected area [2 bytes]

2: word-packed bottom-right y,x of affected area [2 bytes]

4: 8-byte cursor mask [8 bytes]

12: 8-byte cursor sprite [8 bytes]

CHAPTER 6

ASSOCIATED PROGRAM (AP) SYSTEM

The Associated Program (AP) system allows Doors CS to recognize specially-formatted programs as data files with an associated viewer program. When the user runs the data files, Doors CS recognizes them as AP files and instead launches the viewer program, giving it pointers to the data file as necessary. It also handles unarchiving the file before running the viewer and rearchives the files after the viewer closes.

Doors CS contains complex routines for transparently handling archived and unarchived AP files so that the programmer need only write cases for unarchived files. Doors CS also takes care of opening, saving, and saving with a new filename respectively via the FileOpen, FileSave, and FileSaveAs routines. Doors CS dynamically associates files with viewers based on the respective headers of programs on the current calculator, so different viewers can be loaded on a calculator for the same file type. For good examples of Associated Programs in action, take a look at Document DE 7 or mobileTunes v3.0.

The header format for an associated file contains eight bytes:

```
.org datastart
.db $BB,$6D      ;required
.db $C9          ;don't let the TI-OS run this
.db $31,$80      ;this is an AP file
.db 0,1,4        ;3-byte type code (see Registered Filetypes)
Start:
```

Associated programs must follow a variation on the standard Doors CS header for assembly programs that specifies the file type(s) the program can open:

```
; Program Name:
; Author:
; Version:
; Date:
; Written for Doors CS 7.0 and higher (http://dcs.cemetech.net)

.nolist
#include "dcs7.inc"
.list

        .org progstart
        .db $BB,$6D
Init:
        xor d
        ret
        jr Start

        .dw $0000
        .db $07,$00
        .dw Icon
        .dw $0000
        .dw Apstart      ;the routine to open files DCS will start you
                          ;here instead of $9D95 if AP file pending
        .db $31,$7F      ;argh, this be an APMain
        .db $02          ;number of accepted filetypes
MyTypes:
        .db 0,1,4        ;whatever this is
        .db 0,1,2        ;DDE6 text
```

When a user simply runs the specified program from Doors CS, it will start at \$9D95, jump to Start, and begin executing there, as usual. If a user clicks a file meant to be opened with this program, Doors CS will take care of copying the file to RAM if necessary, setting up the viewer program, and then executing it. However, execution will start at APStart instead of at \$9D95. When the program begins executing at APStart, the ix register will contain a pointer to the first data byte of the AP file, ie, the first byte after the 8-byte header. The header will be at ix-8 through ix-1, and the size of the file, including the 8-byte header, will be at ix-10 (LSB) and ix-9 (MSB). The size of the data section of the AP file is of course the file size minus 8.

Within your program, you can modify the AP file as necessary, including inserting memory and deleting memory (be sure to update the size word accordingly!). Make sure you leave the header intact, and only insert or delete memory at ix or higher. When your program quits, Doors CS will writeback the file to the archive if necessary, or delete the temporary RAM copy if it is unchanged. If you call the FileOpen, FileSave, or FileSaveAs within your program, any open AP files will be closed, including removing the RAM copy and writing it back, even if the user chooses not to open a new file. However, if the user chooses not to open a new file, the previously-opened file will be reopened.

Please note that unlike computer programs, in which files must be periodically resaved, Doors CS **does not** require any resaving. Simply update files via _InsertMem and _DeleteMem, and update the size word of the files accordingly, and the file will be updated, and if necessary,

automatically written back to archive by Doors CS. FileSave and FileSaveAs are only necessary to create a file out of raw contents, or to duplicate an existing file.

If you create your own AP-enabled program, be sure to add the filetype to http://dcs.cemetech.net/index.php?title=Registered_Filetypes to avoid conflicts with other programs.

AP Routines

- FileOpen - hooks into the GUI system. Returns pointers to data.
- FileSave - creates a new file given a name, data, and size. GUIless.
- FileSaveAs - hooks into the GUI system. Creates a new file with specified name, data, size, and type.

FileOpen

Description

Opens a GUI to open files. Fully supports the folder system etc.

Technical Details

As of Doors CS 6.4 beta, it uses the new RunProg Chaining system to operate. It puts away any open program before opening a new program, so be aware that pointers in memory might have moved around. It is strongly recommended to re-chkfindsym any RAM (and ROM) variables after calling this function. Note that this function will also move the gui7 AppVar back above any newly-opened files. Doors CS will handle closing any open AP files when your program quits. Note that unlike Doors CS 6.4 beta and lower, Doors CS 6.5 and higher may return pointers to an AP file even if the FileOpen was canceled, ie, the pointers to an already-open file. Nevertheless, it is mandatory that any time FileOpen is called, any open AP file(s) is/are ready to be written back and closed.

Inputs/Outputs/Destroyed

See [FileOpen](#) section in Routine Summary chapter

FileSave

Description

Create a new AP file with a given name, type, size, and data. Operates with no GUI.

Technical Details

As of Doors CS 6.4 beta, it uses the new RunProg Chaining system to operate. It puts away any open program before creating the new program, so be aware that pointers in memory might have moved around. It is strongly recommended to re-chkfindsym any RAM (and ROM) variables after calling this function. Note that this function will also move the gui7 AppVar

back above any newly-created file. Doors CS will handle closing any open AP files when your program quits.

Inputs/Outputs/Destroyed

See [FileSave](#) section in *Routine Summary* chapter

FileSaveAs**Description**

Create a new AP file with a given type, size, and data. Operates with a GUI, used to ask the user for a name for the file, and optionally a folder in which to store the new file.

Technical Details

As of Doors CS 6.4 beta, it uses the new RunProg Chaining system to operate. It puts away any open program before creating the new program, so be aware that pointers in memory might have moved around. It is strongly recommended to re-chkfindsym any RAM (and ROM) variables after calling this function. Note that this function will also move the gui7 AppVar back above any newly-created file. Doors CS will handle closing any open AP files when your program quits.

Inputs/Outputs/Destroyed

See [FileSave](#) section in *Routine Summary* chapter

CHAPTER 7

CALCNET^{2.2}

CALCnet^{2.2} offers a robust, asynchronous end-to-end transport protocol capable of dynamic, stateless network topology reconfiguration, error detection and recovery, and full network features such as arbitrary simultaneous disjoint communication. It supports one-to-one directed transmissions and one-to-many broadcasts to support as wide a set of applications as possible, and is targeted at utilities, chat programs, file-transfer programs, conference applications, and especially multiplayer and massively-multiplayer gaming on TI graphing calculators. As of this publication, the CALCnet^{2.2} protocol has been written into Doors CS 7.1, a shell and GUI developed for calculators by the author. It can be used on TI-83+, TI-83+ Silver Edition, TI-84+, and TI-84+ Silver Edition graphing calculators, as well as via the TI-84+ emulator on the TI-Nspire calculator.

Routines

CALCnet^{2.2} is designed to be relatively easy for programmers to use with minimal complications and training, and to that end exposes only four functions to assembly programmers, two of which need not be used in general programs. The four functions are as follows:

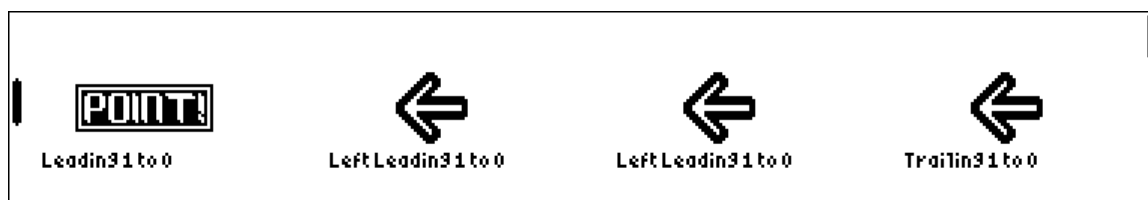
Cn2_Setup - Initialize the CALCnet^{2.2} system, including starting the interrupt and setting up the memory areas and buffers to be used by CALCnet. Once this routine has been called, 547 bytes starting as SavesScreen (\$86EC) will be used by CALCnet and should not be used other than the methods to clear buffers manually or automatically as outlined below. In addition, the RAM segment of the CALCnet^{2.2} interrupt is stored in 42 bytes starting at \$9999 and ending at \$99C3, and the jump table uses 257 bytes from \$9A00 to \$9B01.

Cn2_Setdown - Disabled the CALCnet^{2.2} interrupt, after which the CALCnet buffers can be used again as normal safeRAM.

Cn2_ClearSendBuf - Clears the receive buffer, a total of $256+5+2=263$ bytes. In practice, the receive buffer may be conceptually cleared as far as the interrupt is concerned simply by resetting the MSB of the size field in the buffer, after which you may receive a new frame.

Cn2_ClearRecBuf - Clears the send buffer, a total of $256+5+2=263$ bytes. In practice, the send buffer may be conceptually cleared as far as the interrupt is concerned simply by resetting the MSB of the size field in the buffer, after which you may load in a new frame.

It has been shown that these sets of functions and features are sufficient to create a large set of network-aware games and applications. A screenshot of a sample application that has been released called `NetPong v1.0` can be seen below.



Coding Practices

There is one additional function that CALCnet^{2.2} programs should use; because the TI-OS interrupt that handles GetCSC is disabled while CALCnet is active, programmers should use `call Cn2_GetK` instead. It will debounce the keys entered, so for non-debounced, repeating keypresses like movement in a game, direct input should be used. Programs should also be careful of TI-OS ROM calls (`bcalls`) that disable or enable interrupts, such as `_RunIndicOff` and `_RunIndicOn`, and use alternatives for such routines. The CALCnet^{2.2} interrupt will handle turning the calculator off and on via the **[ON]** key, so user programs need not provide this functionality. Finally, be aware that during the CALCnet interrupt every calculator, regardless of type, will be running in 6MHz mode. Programs that require 15MHz mode are recommended not to simultaneously use 15MHz mode and CALCnet^{2.2}: although the interrupt will properly handle transitioning to 6MHz mode for its duration and returning to 15MHz mode before the user program resumes execution, 15MHz mode will cause the CALCnet interrupt to trigger at 275Hz instead of 110Hz, thus spending proportionally more time in the interrupt than in the user program, and possibly nullifying the objective of 15MHz mode (ie, faster execution of the user program).

Memory Areas

As mentioned previously, CALCnet^{2.2} relies on the use of buffers to send and receive data. Programs need not call any routines in order to initiate transmission or reception of data, allowing for asynchronous, nonblocking communication. This means that a program can leave data to be sent over the network in a buffer and continue to execute; CALCnet will handle attempting to send the specified data until it succeeds. Similarly, the interrupt listens constantly (technically, 110 times per second) for data to be sent to the calculator, and will store any such data in a buffer until fetched by the userland program. Needless to say, it is important to periodically check for received data and remove it so that CALCnet can accept another frame. The memory areas used by CALCnet are enumerated in the table below and are further enumerated in the remainder of this section.

Address	Offset	Size	Function
\$86EC	0	2	<i>not for userland user</i>
\$86EE	2	5	Current calc's ID (do not modify)
\$86F3	7	5	Receive buffer sender ID
\$86F8	12	2	Receive buffer size word
\$86FA	14	256	Receive buffer
\$87FA	270	5	Send buffer receiver ID
\$87FF	275	2	Send buffer size word

\$8801	277	256	Send buffer
\$8901	533	15	<i>not for userland use</i>

Caveats

It has been found that there are several TI-OS related caveats of which to be aware when using CALCnet^{2.2}. All such caveats are due to the interactions between the TI-OS and user interrupts (Interrupt Mode 2 ISRs, such as CALCnet). It has been found that certain TI-OS bcalls, especially those that potentially interact with the LCD, maybe improperly disable interrupts including the CALCnet ISR. Known culprits including `_vputs` and `_vputmap`, but other ROM calls are likely to cause problems. It is strongly recommended that programs that using the CALCnet^{2.2} ISR re-enable interrupts via `ei` after one or more of such calls. If there are sections of your code that may or may not have interrupts enabled, the following construction may be used (thanks to Brenden "Calc84Maniac" Fletcher for his assistance):

```

;begin protected section
xor a
push af
    pop af
ld a,i
jp pe,sectionbody
dec sp
dec sp
pop af
add a,a
jr z,sectionbody
xor a
sectionbody:
    push af
        ;di ;uncomment if interrupts should be disabled here
        ;-----code that may have disable interrupts or-----
        ;-----need interrupts disabled goes here-----
    pop af
    jp po,sectionend
ei
sectionend:

```

Finding Network Members

CALCnet^{2.2} does not contain an explicit method of finding network members; the send and receive routines were initially designed to be purely point-to-point. Programs that already know the 5-byte address of the receiving calculator can simply fill in that address in a frame to be sent, and the data will arrive at the receiver. However, for dynamic networks, it will be necessary for a program to find other calculators in the network. In order to do so, each calculator can be set to periodically broadcast its address to other calculators during a discovery phase or even as a packet type during normal operation. Because CALCnet^{2.2} only defines at most the bottom three layers of the OSI model (physical protocol, the data layer, and the physical layer), it is up to each program to define the meaning of the contents of the

data section of frames. However, if a program puts a frame in its calculator's CALCnet send buffer with a 5-byte receiver address of 000000, CALCnet will interpret that address as indicating a *broadcast frame*, and will send that frame to every calculator on the network. Note that broadcast transmission is somewhat less reliable than point-to-point transmission. Whereas point-to-point guarantees that once the data disappears from the sender, the receiver has successfully received and acknowledged the data, successful transmission of a broadcast frame does *not* guarantee that every or even any calculator has received the data. Repeated broadcast transmission of the broadcast packet should offer a reasonable chance that each calculator will receive the packet, but the broadcast should not be continually attempted to avoid monopolizing the network.

Sending

To send data via CALCnet^{2.2}, a calculator program must first place the receiver ID, the frame contents, and the size of the frame to be transmitted in the associated memory areas, in that order (see the table above). The order is important because CALCnet notes whether a frame is pending transmission by checking the most significant bit of the two-byte send buffer size word. Because the z80 follows the little-endian storage model (the less significant byte of a word is stored before the more significant byte), the most significant bit is bit 7 of the byte at \$8800 (see the table above if the derivation of this address is not clear). If that most significant bit is reset, the CALCnet^{2.2} interrupt assumes that there is no frame pending transmission. Once the bit is set, CALCnet may begin transmitting the frame at any point thereafter. Due to the asynchronous nature of interrupts, the interrupt may fire at any time, so it is vital that the high bit of the size word must only be set when the low size byte, receiver ID, and frame data contents have already been written. To facilitate direct copying into the buffer from RAM or ROM via `ldir` or an equivalent, the size may be copied with the MSB unset, then the MSB of the size word's high byte should be set. The biggest danger of prematurely setting the bit and thus prematurely indicating to the CALCnet interrupt that a frame is pending transmission is that a partially-complete frame may be transmitted. Testing and verification shows that this can happen frequently under real-world operation if the caveats dictated in this paragraph are not followed.

Once a completed frame has been placed in the send buffer, the interrupt may take a number of seconds to transmit the given frame in the range $[\epsilon, \infty)$, in practice roughly $[0.01, \infty)$ seconds. Transmission will take a finite amount of time if the receiving calculator is on the network and the network is sufficiently non-noisy and has low enough end-to-end latency to ensure proper transmission. A back-of-the-envelope calculation, taking the minimum granularity of network time as the unrealistically strict value $>10\mu\text{s}$, dictates that a maximum transmission distance based on the speed of light is 29979 meters, or 30 kilometers or 18 miles. Therefore, any transmission over a wide-area network should be performed via an intermediary, such as the planned **globalCALCnet** (gCn) system. If the receiving calculator is not on the network, for example if the receiver's ID was incorrectly entered into the associated field in CALCnet

send buffer, then the transmission will take an infinite amount of time, as the remote calculator will never acknowledge the packet. Transmitting calculators should provide a means of removing a frame from the CALCnet send buffer if it has been pending for a long time and has not been send successfully; it is up to the programmer to decide how to do this. One recommended method follows in the code listing below:

```
xor a
halt                ;halt to ensure that the store
ld (8800h),a        ;happens immediately after an interrupt
```

A program will almost definitely need to know when it is safe to load a new frame into the CALCnet send buffer. This can be done via the same method that the interrupt itself uses, ie, loading the byte at (8800h), masking off the most significant bit, and loading a new frame if and only if that bit is reset (zero). One note: the MSB is not considered part of the size, so that a size of \$8002 is considered a 2-byte frame, not a 32770-byte frame.

Receiving

Receiving a packet carries similar caveats and complexities as transmission. As with transmission, there is a single bit to read and write to indicate the presence or absence of pending data. As with transmission, there are three field in the relevant buffer: the two-byte size of the data (for which a value of \$8004 is considered a 4-byte frame, not a 32772-byte frame), the five-byte ID of the sending calculator, and the 1- to 255-byte data section. As with transmission, broadcast frames must be handled, although from the receiver side, a broadcast frame is indistinguishable from a normal directed (point-to-point) frame. Finally, in symmetry to the transmission routine, the user program must clear the present-data bit when it has read the data out of the receive buffer in order to allow the CALCnet^{2.2} interrupt to accept another frame.

As with transmission, the memory areas relevant to receiving data and their locations and sizes are in the table above. The receive buffer will contain a size word indicating the size of the data in the buffer; the data is only valid if the highest bit of the size word is set. CALCnet uses the receive buffer to store frames currently in transit to this calculator before each frame's checksum is verified, so only when the high bit of the size word is set should the data be read. Once the receive buffer is used, the CALCnet^{2.2} will ignore further frames transmitted to this calculator until the receive buffer is cleared, as indicated by the high bit of the size word being reset, so to maintain a low-latency network, the receive buffer should be checked and handled as often as possible.

More Information

Additional information about the CALCnet^{2.2} protocol can be found in the CaLCnet^{2.2} Whitepaper, available here:

<http://www.cemotech.net/news.php?year=2010&month=all&id=419>

You may also request assistance on the Cemotech forum in the CALCnet section:

<http://www.cemotech.net/forum/viewforum.php?f=67>

CHAPTER 8

SHELL EXPANSIONS

Shell Expansions, or SEs, are modules for Doors CS. They are similar to the concept of Ion Modules, except that they are far more versatile. Shell Expansions can have one or more of three sections: Boot, Mouse, and Unload. A shell expansion with a Boot section is called while Doors CS is booting, before it counts programs and displays the desktop. Possible uses: load interrupts, set up hardware, sort programs, etc etc. Mouse sections are the most versatile: not only does Doors CS call them continuously while the mouse is active, they can actually interject actions into Doors CS itself. They can force mouse movement, clicking, and coordinate jumping via well-placed memory editing (not necessarily advocated). Finally, unload sections are, obviously, called while Doors CS is quitting, after it has closed down the GUI system but before it reverts to the TI-OS. Since a program can contain one or more of these sections, it can be anywhere from 768 byte to 2,304 bytes; each section can only be up to 768 bytes, since it must fit in SavesScreen.

Why Write a Shell Expansion?

As a programmer, why would you want to write your program as an SE rather than a full program? Well, not only would you have the programming challenge of fitting your entire program into 768, 1,536, or 2,304 bytes, you would have the advantage of interacting directly with the Doors CS kernel. Startup programs could directly set flags and options within Doors CS, or through some fancy programming, even prevent it from booting entirely (password protection, anyone?). Because Doors CS is now an app, SEs have access to each and every routine Doors CS has to offer to ASM programs, although it is not recommended that they mess with the GUI system, in case Doors CS itself is taking advantage of undocumented GUI API features. The mouse SEs are the crowning jewel of the SE system, for they allow the programmer full control over cursor movement, cursor clicking, and even the direct cursor coordinates. Such power gives SEs the opportunity to completely change the way the interface works for the user. A low-end example could be remapping the keys on the keyboard, while a higher-end one might be a PS/2 or USB hardware driver.

Strengths and Weaknesses of SEs

Shell Expansions have several strengths and weaknesses that you need to take into account when designing one. Shell Expansions can:

- Call any TI-OS ROM call
- Read data from any memory location
- Use any register (no need to shadow)
- Jump relative within the SE

- Jump absolute within the SE (see below for restrictions)

However, there are some things that Shell Expansions *cannot* or *should not* do:

- Jump absolute outside of the SE (except to a ROM call)
- Exceed 768 bytes per section, or 2304 bytes total
- Write back to the program for later use
- Create or delete programs or data in such a way that the VAT will be disturbed
- Use SafeRAM4

Shell Expansions can jump absolutely within the SE, but instead of the usual format, absolute jumps must be taken this way:

```
SELoad:
    prog commands
    here
    jp z,Label-SELoad+$8265
    more commands
    ret
Label:
    Commands
    ret
```

Alternatively, of course, you can use your assembler of choice's `.org`, `.rorg`, or `.reorg` directive to achieve the same end.

Technical Details

Persistent Storage

Each SE is allocated two bytes of RAM for mode storage. IT can be found at (ix) and (ix+1) when the SE is loaded.

dModes: Return Values From SEs

These are the values that are returned from the Mouse section of any SE, and represent an action that the SE would like Doors CS to perform.

- 0 = nothing
- 1 = leftclick
- 2 = rightclick
- 3 = up
- 4 = down
- 5 = left
- 6 = down
- 7-102 = set X-coord to d-7

- 103-166 = set Y-coord to d-103

In some betas between Doors CS 6.2 and Doors CS 6.7, notably 6.3, 6.4, 6.5, and 6.6, the dModes listed above were nonfunctional.

Shell Expansion Header

Like other types of Doors CS programs and files, Shell Expansions must use a special SE header.

```
.nolist                                     ;defines, includes, and equates
#include "dcs7.inc"                         ;token interpretation
SEOffset      equ      $86ec
.list

      .org 0
      .db $BB,$6D                          ;ASM program header
      .db $AB,$C9                          ;ignore program; ret
SEStart:
      .db $31,$87,$50                      ;SE for DCS 5 and higher
      .dw SEBoot-SEStart                   ;or .dw 0 if there is no boot section
      .dw SEMouse-SEStart                  ;or .dw 0 if there is no mouse section
      .dw SEUnload-SEStart                 ;or .dw 0 if there is no unload section
SEBoot:                                ;if there is a boot section...
      ;...code...
      ret
SEMouse:                                ;if there is a mouse section...
      ;...code...
      ld d,0
      ret
SEUnload:                                ;if there is an unload section...
      ;...code...
      ret
```

CHAPTER 9

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:

- [Ion Compatibility Routines](#)
- [Doors CS Legacy Routines](#)
- [Doors CS Graphical User Interface \(GUI\) Routines](#)
- [CALCnet^{2.2} Routines](#)
- [Doors CS Associated Program \(AP\) Routines](#)
- [Mirage OS Compatibility Routines](#)

Ion Compatibility Routines

iVersion

<http://dcs.cemetech.net/index.php?title=IVersion>

Description

Returns the version of the Ion libraries supported by Doors CS.

Technical Details

Doors CS emulates the libraries of Ion v1.6, containing 8 library routines.

Inputs

None

Outputs

a = Ion compatibility number (0)

d = Library compatibility number (0)

e = Number of supported routines (8)

h = Major Ion version emulated (1)

I = Minor Ion version emulated (6)

Destroyed

a, de, hl

iRandom

<http://dcs.cemetech.net/index.php?title=IRandom>

Description

Returns a random number within set bounds, such that the returned value a is, in relation to the input upper bound b: $0 \leq a < b$

Technical Details

Inputs

b = One larger than the highest random number this routine may return

Outputs

a = A random number between 0 and b-1, inclusive.

b = 0

Destroyed

All

iPutSprite

<http://dcs.cemetech.net/index.php?title=IPutSprite>

Description

XORs an 8-pixel-wide sprite to the graph buffer, aligned or unaligned. Does not clip. The sprite can be any height 1-64 pixels.

Technical Details**Inputs**

a = X-coordinate of upper-left of sprite

b = Sprite height, in pixels

l = Y-coordinate of upper-left of sprite

ix = Pointer to first byte of sprite

Outputs

Sprite XORed to graph buffer

Destroyed

All

iLargeSprite

<http://dcs.cemetech.net/index.php?title=ILargeSprite>

Description

Draws an aligned or unaligned sprite to the graph buffer with a width that is a multiple of 8 pixels, and any height. Does not perform clipping.

Technical Details**Inputs**

a = X-coordinate of top-left of sprite

b = Sprite height

c = Sprite width (in bytes, so divide by 8)

l = Y-coordinate of top-left of sprite

ix = Pointer to first byte of sprite

Outputs

Draws the sprite to the graph buffer.

Destroyed

All

iFastCopy

<http://dcs.cemetech.net/index.php?title=IFastCopy>

Description

Copies the graph buffer to the LCD, very fast. Properly handles the emulator in the Nspire as well as the delays necessary for the TI Presenter device.

Technical Details

This routine temporarily disables interrupts, but is careful to re-enable them when it completes.

Inputs

None

Outputs

Graph buffer copied to LCD.

Destroyed

All

iDetect

<http://dcs.cemetech.net/index.php?title=iDetect>

Description

Finds a give file in the VAT, if it exists, searching based on the contents of the file. This routine is designed to be reentrant, in the sense that you can call it multiple times with inputs based on its outputs. For example, you can call it the first time with hl=(progptr), and then subsequent times with hl = the de output of the previous run. It can also properly deal with archived programs. This is particularly useful for finding game levels that always have a specific header (or even DCS APs!).

Technical Details**Inputs**

hl = Pointer to where to begin in the VAT, for example (progptr)

ix = Detection string, zero-terminated

Outputs

de = Address of place stopped in the VAT + 1

hl = Pointer to first byte of program data

z flag = set if no program was found (hl and de are invalid), and reset if a program was found

Destroyed

af, bc, de, hl

iDecompress

<http://dcs.cemetech.net/index.php?title=iDecompress>

Description

Decompresses bitpacked data.

Technical Details**Inputs**

b = length of compressed data.

c = compression factor, either 1, 3, or 15.

hl = Pointer to first byte of compressed data.

de = Pointer to first byte of RAM to store decompressed data.

Outputs

Data decompressed from hl to de.

hl = Pointer to next byte of compressed data.

Destroyed

af, bc, de, hl

iGetPixel

<http://dcs.cemetech.net/index.php?title=IGetPixel>

Description

Gets the offset of a specified pixel into the graph buffer, as well as a bitmask isolating the selected pixel in its group of eight.

Technical Details**Inputs**

a = x-coordinate of pixel

e = y-coordinate of pixel

Outputs

a = pixel mask for specified pixel

hl = address of 8-pixel group containing this pixel in the graph buffer

Destroyed

All

Doors CS Legacy Routines

SmallWindow

<http://dcs.cemetech.net/index.php?title=SmallWindow>

Description

This routine previously drew a centered small window on the screen in Doors CS 5.0 and lower. It is **strongly** recommended that this routine not be used, and instead the power of the GUI API be harnessed.

SmallWindowErase

<http://dcs.cemetech.net/index.php?title=SmallWindowErase>

Description

This routine previously erased a SmallWindow-sized area centered on the LCD. It is very strongly recommended that it not be used, and that the GUI API be used instead.

LargeWindow

<http://dcs.cemetech.net/index.php?title=LargeWindow>

Description

This routine previously drew a full-screen large window in Doors CS 5.0 and lower. It is **strongly** recommended that this routine not be used, and instead the power of the GUI API be harnessed.

LargeWindowErase

<http://dcs.cemetech.net/index.php?title=LargeWindowErase>

Description

This routine previously erased a LargeWindow, ie, cleared the screen. It is very strongly recommended that the GUI API be used instead of this and related functions.

PlaySound

<http://dcs.cemetech.net/index.php?title=PlaySound>

Description

Plays a monophonic note on both channels of the serial I/O port, which can be heard with either headphones or an AM radio tuned to static.

Technical Details

Inputs

hl = frequency of note (see notes.inc in the Doors CS 7.0 SDK)

de = duration of note (see notes.inc in the Doors CS 7.0 SDK)

Outputs

The specified note is played. Note that this routine is blocking: ie, it will not return to the caller program until the end of the note.

Destroyed

All

VDispHL

<http://dcs.cemetech.net/index.php?title=VDispHL>

Description

Displays HL as an up to five-digit positive integer onscreen.

Technical Details**Inputs**

hl = value to display

Outputs

Up to five digits of HL displayed onscreen starting at (pencol, penrow).

Destroyed

af, hl, Op1, Op2

Pause

<http://dcs.cemetech.net/index.php?title=Pause>

Description

Waits for a keypress, then returns the key code _getcsc-style in a.

Technical Details**Inputs**

None

Outputs

Returns keycode in a. Zero is not a valid output from this function.

Destroyed

af

hDetect

<http://dcs.cemetech.net/index.php?title=HDetect>

Description

This routine checks if a coordinate pair is within a rectangular area, the top-left and bottom-right of which are defined by two other coordinate pairs. It can be used for collision detection or mouse click evaluation, among other things.

Technical Details**Inputs**

hl = Location to check if within; (h,l)

bc = Upper left corner of area (b,c)

de = Lower right corner of area (d,e)

Outputs

Zero flag is set if hl is inside area, reset otherwise.

Destroyed

af, bc, de, hl

DispLongInt

<http://dcs.cemetech.net/index.php?title=DispLongInt>

Description

Displays an integer between 0 and 16,777,215 (inclusive) onscreen in small (graphscreen) font.

Technical Details**Inputs**

hl = pointer to the MSB of a three-byte big-endian integer

pencol/penrow = location to display number

Outputs

Up to 8 digits displayed onscreen

Destroyed

af, bc, de, hl, Op1

DPutMap

<http://dcs.cemetech.net/index.php?title=DPutMap>

Description

Display a character using the Doors CS font-display routines, completely separate from (and

somewhat faster than) the TI-OS VPutMap routine. In the future it will support multiple font sets, although currently only fontset 0 (3x5 variable-width graphscreen font) is supported.

Technical Details

Inputs

a = character to display

e = fontset to use (only set 0 is currently valid)

Outputs

Character displayed onscreen at (pencol, penrow) and pencol advanced by the width of the character including its rightmost blank column.

Destroyed

af, bc, de, hl

AP_GUIAVToTop

http://dcs.cemetech.net/index.php?title=AP_GUIAVToTop

Description

This internal Doors CS routine is used to move the "gui7" AppVar to the highest spot in user memory (directly below the free RAM chunk) so that the size of the appvar can change without needing to move any data above it. Although this is primarily used by Doors CS itself, it should be called if you manually create any files from within your program. If you use FileOpen, FileSave, or FileSaveAs, you need not call this routine, as Doors CS will handle it automatically.

Technical Details

Inputs

None

Outputs

gui7 moved above any newly-created programs or files

Destroyed

All

RunProg

<http://dcs.cemetech.net/index.php?title=RunProg>

Description

Runs a program via Doors CS's RunProg subsystem. This call can handle any program that Doors CS itself can handle, including AP programs and AP files, nostub BASIC and Hybrid BASIC, Axe source code, nostub, DCS, Ion, and MirageOS ASM, plus any other filetype that

gets added to Doors CS 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

Doors CS Graphical User Interface (GUI) Routines

OpenGUIStack

<http://dcs.cemetech.net/index.php?title=OpenGUIStack>

Description

Creates and initializes the Doors CS GUI system. Sets up all appvars and variables necessary. Repeatedly calling this function will have no negative affects, but the c register will be modified to indicate whether the GUI stack had already been opened previously. The counterpart to this function is the CloseGUIStack routine. Note that the GUI uses several SafeRAM areas that are normally reserved for ASM programs, so your code should be written accordingly.

Technical Details

Inputs

a = \$c7 to use special VAT-swapping speed optimization. If not \$c7, the optimization will be omitted. In general, it is safe to avoid explicitly setting the accumulator. It is not recommended that programs use the \$C7 mode, as it will offset any pointers into the VAT by 11 bytes.

Outputs

c = 0 if the stack was closed, 1 if the stack was open.

hl = first data byte of the stack appvar

de = first byte of the VAT data

Shortcut

OpenGUIStack()

Destroyed

a, bc, de, hl, Op1

PushGUIStack

<http://dcs.cemetech.net/index.php?title=PushGUIStack>

Description

Pushes an item onto the GUI stack. The first element on the stack must be either GUIRNull, GUIRLargeWin, or GUIRSmallWin. Every time one of those three items appears on the stack, it indicates the start of a new entry. This function takes three arguments. Note that the OpenGUIStack routine should be called before using this function. Also note that if there is insufficient RAM to fit the GUI elements being pushed, undesirable results may be obtained. Doors CS will check for and catch situations in which there is less memory than would be required for a push operation, but the program initiating the push(es) should be aware that it

may lose control in such circumstance. Avoid pushing excessive information onto the GUI Stack without checking that there is sufficient memory available to do so. Also be aware that a very small amount of data is added to the GUI Stack within the GUIMouse routine to incorporate clickable hotspots.

Technical Details

Inputs

a = Item type. Examples: GUIRSmallWin, GUIRRadio, GUIRPassIn

hl = Pointer to stack entry data.

de = Length of data in bytes.

Outputs

Adds the specified entry to the GUI stack. This routine does not render the GUI, however.

hl = first byte of stack entry (after 2 bytes of size and 1 byte of type)

Shortcut

PushGUIStack([a],[hl],[de])

Destroyed

All

PushGUIStacks

<http://dcs.cemetech.net/index.php?title=PushGUIStacks>

Description

Pushes a group of items onto the GUI stack.

Technical Details

The data chunk at hl consists of several entries to insert into the GUI stack, each formatted as:

```
.dw Size  
.db Type  
.db Data
```

Please note that the Size is the size of the data plus the type byte plus the size word, or `sizeof(data)+3`. After the last item, the two-byte sequence `$FF,$FF` must be appended to signal the end of the data section.

Inputs

hl = Pointer to first byte of stack data.

Outputs

Adds the specified entries to the GUI stack. This routine does not render the GUI, however.

Shortcut

PushGUIStacks([hl])

Destroyed

All

GUIFindFirst

<http://dcs.cemetech.net/index.php?title=GUIFindFirst>

Description

Finds the first non-groupmaster element in the 'top group' of the GUI Stack. In other words, it actually returns the second item in the group, after the GUIRSmallWin, GUIRLargeWin, or GUIRnull item. Returns an error if there is no open GUI stack. This routine is useful for extracting information from input forms after the user quits a form or other GUI group.

Technical Details**Inputs**

None

Outputs

c = 0 if the stack was open and this routine succeeded; 1 if the stack was closed, therefore no items to see.

hl = the first byte of the size of the first element in the top group

de = the byte after the last byte of the top group.

It is recommended that you push hl and de if you plan to use GUIFindNext; that routine requires the data of these two outputs as inputs.

Shortcut

GUIFindFirst()

Destroyed

a, bc, de, hl, Op1

GUIFindNext

<http://dcs.cemetech.net/index.php?title=GUIFindNext>

Description

Finds the next element in the 'top group' of the GUI Stack. It assumes you have already called GUIFindFirst or have in some way verified that the stack is actually open (via OpenGUIStack) and elements are on it (via PushGUIStack). This routine is useful for extracting information from input forms after the user quits a form or other GUI group.

Technical Details

Inputs

hl = the first byte of the entry the program is currently at.

de = the first byte after the last byte of the top group.

You can use the hl and de values that were returned by GUIFindFirst or a previous call of GUIFindNext

Outputs

c = 0 if there is a next item and de and hl are valid; 1 if we were already at the end and hl / de are invalid.

hl = the first byte of the size of the next element in the top group

de = the byte after the last byte of the top group.

It is recommended that you push hl and de if you plan to use GUIFindNext again; that routine requires the data of these two outputs as inputs.

Shortcut

GUIFindNext([hl],[de]) - To specify non-register values

GUIFindNext() - To use already-set contents of hl and de

Destroyed

a, bc, de, hl, Op1

GUIFindThis

<http://dcs.cemetech.net/index.php?title=GUIFindThis>

Description

Finds the ath non-groupmaster element in the 'top group' of the GUI Stack. In other words, it actually returns the second or later item in the group, after the GUIRSmallWin, GUIRLargeWin, or GUIRnull item. Unlike GUIFindFirst and GUIFindNext, assumes there is an open GUI stack. This routine is useful for extracting information from input forms after the user quits a form or other GUI group.

Technical Details**Inputs**

a = number of items to skip before returning an item

Outputs

hl = the first byte of the size of the first element in the top group

de = the byte after the last byte of the top group.

bc = size of the current element

It is recommended that you push hl and de if you plan to use GUIFindNext; that routine requires the data of these two outputs as inputs.

Shortcut

GUIFindThis([a])

Destroyed

a, bc, de, hl, Op1

RenderGUI

<http://dcs.cemetech.net/index.php?title=RenderGUI>

Description

This routine renders the GUI to the graph buffer, then copies the graph buffer to the screen. It renders every item on the stack, layering multiple entries if necessary.

Technical Details**Inputs**

None

Outputs

Draws the GUI to the LCD

Shortcut

RenderGUI()

GUIRender()

Destroyed

All

GUIMouse

<http://dcs.cemetech.net/index.php?title=GUIMouse>

Description

Renders the contents of the GUIStack and begins input functions.

Technical Details

Please note three issues when calling GUIMouse:

1. RenderGUI is automatically called from GUIMouse. If you're going to be using GUIMouse, don't bother with RenderGUI first.
 2. You need to call GUIMouse, but when control is returned to your program, the stack will be in a condition as if you jumped to GUIMouse. Keep that in mind!!
 3. You must call ResetAppPage as the very first instruction anywhere in your program to which GUIMouse might jump.
-

Inputs

hl = Location of hook to call once per cycle. You must set this to \$0000 if not using one. Hook Note: the current mouse location as (x,y) will be in (a,l) when the hook routine triggers. You can also get/set the location by checking in MouseX and MouseY. To force the GUIMouse to return control to your program from inside the hook, set the coordinates to an object that would cause it to exit anyway, and add an extra pop to the routine. if you have no suitable gui object you can use an offscreen GUIRHotspot, and set the mouse location to that.

if the hook begins with .db \$3f it will disable the debouncing routines, the click checking, and use a custom acceleration mode. This option is useful if you need to be able to press keys while moving the mouse.

Outputs

May update data inside elements on the GUI Stack. Will end up jumping to the location defined by hotspots, etc.

Shortcut

GUIMouse() - Use this if there's no hook

GUIMouse([hl]) - Pass a label name of the hook

Destroyed

All plus GUI Memory Areas

PopGUIStack

<http://dcs.cemetech.net/index.php?title=PopGUIStack>

Description

Removes the top item from the GUIStack. If the GUI stack was empty, nothing happens.

Technical Details**Inputs**

None

Outputs

bc = number of bytes removed. Can be used for ptr updates.

hl = where it was removed from.

Shortcut

PopGUIStack()

Destroyed

af, bc, de, hl, Op1

PopGUIStacks

<http://dcs.cemetech.net/index.php?title=PopGUIStacks>

Description

Pop one or more items from the top of the GUIStack. If the programmer requests more items be popped than currently exist on the stack, then the stack will be emptied and then the routine will return.

Technical Details

Inputs

b = Number of items to pop from the stack

Outputs

b = 0

Shortcut

PopGUIStacks([b]) - Pass an integer indicating how many to pop

Destroyed

All but c

CloseGUIStack

<http://dcs.cemetech.net/index.php?title=CloseGUIStack>

Description

Closes the Doors CS GUI system and removes the gui6 appvar from memory. No ill effects can be accidentally incurred by calling this routine multiple times, although the return value of the c register is used to indicate whether the stack had already been closed. This routine is the counterpart of the OpenGUIStack function. Warning: this will delete any information currently in the GUI stack. Copy any important data into files or SafeRAM before closing the stack.

Technical Details

Inputs

None

Outputs

c = 0 if the stack was open, now close; 1 if the stack was already closed.

hl = size of stack before close if c=0.

de = where stack was if c=0.

(Previous two register values useful for ptr updating)

Shortcut

CloseGUIStack()

Destroyed

a, bc, de, hl, Op1

ResetAppPage

<http://dcs.cemetech.net/index.php?title=ResetAppPage>

Description

Resets Doors CS's active page to page 0.

Technical Details

This routine must be called as the first command of a segment of code triggered by GUIMouse.

Inputs

none

Outputs

none

Shortcut

ResetAppPage()

MouseRecover() - Alias to ResetAppPage()

Destroyed

hl, de, bc

CALCnet^{2,2} Routines

Cn2_Setup

http://dcs.cemetech.net/index.php?title=Cn2_Setup

Description

Initialize the CALCnet system, including starting the interrupt and setting up the memory areas and buffers to be used by CALCnet. CALCnet is a linking/networking library for linking between two and 16¹⁰ calculators or other two-wire serial devices together in a decentralized network.

Technical Details

Inputs

None

Outputs

The CALCnet^{2,2} interrupt is installed. Be sure to carefully read the CALCnet^{2,2} chapter to understand the memory areas that will be used and the routines that must be used and avoided.

Destroyed

All

Cn2_Setdown

http://dcs.cemetech.net/index.php?title=Cn2_Setdown

Description

Disable the CALCnet system, including shutting down the interrupt and freeing the memory areas used by CALCnet for the interrupt and its buffer.

Technical Details

Inputs

None

Outputs

The CALCnet^{2,2} interrupt is disabled and its memory areas are freed.

Destroyed

All

Cn2_ClearRecBuf

http://dcs.cemetech.net/index.php?title=Cn2_ClearRecBuf

Description

Clears the receive buffer, a total of $256+5+2=263$ bytes. In practice, the receive buffer may be conceptually cleared as far as the interrupt is concerned simply by resetting the MSB of the size field in the buffer, after which you may receive a new frame.

Technical Details**Inputs**

None

Outputs

Receive buffer is cleared. See caveat in Description above.

Destroyed

All

Cn2_ClearSendBuf

http://dcs.cemetech.net/index.php?title=Cn2_ClearSendBuf

Description

Clears the send buffer, a total of $256+5+2=263$ bytes. In practice, the send buffer may be conceptually cleared as far as the interrupt is concerned simply by resetting the MSB of the size field in the buffer, after which you may load in a new frame.

Technical Details**Inputs**

None

Outputs

Send buffer is cleared. See caveat in Description above.

Destroyed

All

Doors CS AP Routines

FileOpen

<http://dcs.cemetech.net/index.php?title=FileOpen>

Description

Opens a GUI to open files. Fully supports the folder system etc.

Technical Details

As of Doors CS 6.4 beta, it uses the new RunProg Chaining system to operate. It puts away any open program before opening a new program, so be aware that pointers in memory might have moved around. It is strongly recommended to re-chkfindsym any RAM (and ROM) variables after calling this function. Note that this function will also move the gui7 AppVar back above any newly-opened files. Doors CS will handle closing any open AP files when your program quits. Note that unlike Doors CS 6.4 beta and lower, Doors CS 6.5 and higher may return pointers to an AP file even if the FileOpen was canceled, ie, the pointers to an already-open file. Nevertheless, it is mandatory that any time FileOpen is called, any open AP file(s) is/are ready to be written back and closed.

Inputs

a = zero to only show files of the type(s) defined in this AP's header, or nonzero to show all oncalc files

Outputs

hl = 0 if no file was returned, or the first byte of data if a file was returned

de = pointer to the size word of the data for updating. Please note: the size at the size word is not the same as the size word in bc. bc is 8 bytes smaller than (de) due to the AP header (\$BB,\$6D,\$C9,\$31,\$80,type,type,type)

bc = size of the pure data section of the file.

If necessary, you should also check hl-3 through hl-1 (3 bytes) to see if it's a valid type for this program.

Destroyed

a, bc, de, hl, Op1, iMathPtrs, GUI Memory Areas

FileSave

<http://dcs.cemetech.net/index.php?title=FileSave>

Description

Create a new AP file with a given name, type, size, and data. Operates with no GUI.

Technical Details

As of Doors CS 6.4 beta, it uses the new RunProg Chaining system to operate. It puts away any open program before creating the new program, so be aware that pointers in memory might have moved around. It is strongly recommended to re-chkfindsym any RAM (and ROM) variables after calling this function. Note that this function will also move the gui7 AppVar back above any newly-created file. Doors CS will handle closing any open AP files when your program quits.

Inputs

hl = pointer to data to put in new file

bc = size of pure data section (the program created will be larger than this because of the AP header)

de = pointer to 3-byte type sequence

op1 contains name, eg. .db 06,"NAME",0

Outputs

zero flag = Set if successful. If not zero, then the function FAILED, and the existing open file might have been closed! Be aware of this.

hl = 0 if no file was returned, or the first byte of data if a file was returned

de = pointer to the size word of the data for updating. Please note: the size at the size word is not the same as the size word in bc. bc is 8 bytes smaller than (de) due to the AP header (\$BB,\$6D,\$C9,\$31,\$80,type,type,type)

bc = size of the pure data section of the file.

Destroyed

a, bc, de, hl, Op1, iMathPtrs, GUI Memory Areas

FileSaveAs

<http://dcs.cemetech.net/index.php?title=FileSaveAs>

Description

Create a new AP file with a given type, size, and data. Operates with a GUI, used to ask the user for a name for the file, and optionally a folder in which to store the new file.

Technical Details

As of Doors CS 6.4 beta, it uses the new RunProg Chaining system to operate. It puts away any open program before creating the new program, so be aware that pointers in memory might have moved around. It is strongly recommended to re-chkfindsym any RAM (and ROM) variables after calling this function. Note that this function will also move the gui7 AppVar back above any newly-created file. Doors CS will handle closing any open AP files when your program quits.

Inputs

a = zero to only show files of the type(s) defined in this AP's header, or nonzero to show all oncalc files
hl = pointer to data to put in new file

bc = size of pure data section (the program created will be larger than this because of the AP header)

de = pointer to 3-byte type sequence

Note that unlike FileSave, Op1 is not used as an input (although it is destroyed in the course of fileSaveAs execution)

Outputs

zero flag = Set if successful. If not zero, then the function FAILED, and the existing open file might have been closed! Be aware of this.

hl = 0 if no file was returned, or the first byte of data if a file was returned

de = pointer to the size word of the data for updating. Please note: the size at the size word is not the same as the size word in bc. bc is 8 bytes smaller than (de) due to the AP header (\$BB,\$6D,\$C9,\$31,\$80,type,type,type)

bc = size of the pure data section of the file.

Destroyed

a, bc, de, hl, Op1, iMathPtrs, GUI Memory Areas

MirageOS Routines

Please see the MirageOS SDK (<http://www.detachedsolutions.com/mirageos/develop/>) for the full listing of inputs and outputs for these routines. A summary of the available routines in Doors CS ported from MirageOS can be found below. Note that if a routine is not listed here, it is **not** available, even if it's in the MirageOS SDK.

- **directin** - scans one group of the keypad for pressed keys via direct input.
- **sendbytetios** - sends one byte via TI-OS linking routines.
- **getbytetios** - receives one byte via TI-OS linking routines.
- **version** - retrieves MirageOS library compatibility information.
- **setvputs** - sets (penrow) and (pencol) and displays a string in variable-width graphscreen font.
- **setpixel** - sets a specific pixel in the graph buffer to dark.
- **fastcopy** - copies the graph buffer to the LCD, preserving all registers.
- **delayb** - halts for a specified amount of time.
- **multhe** - multiplies h*e.
- **multhl** - multiplies h*l.
- **quittoshell** - quits directly to Doors CS, regardless of stack level.
- **fastline** - draws a line to the graph buffer, fast.
- **pixelonhl** - similar to setpixel
- **pixeloff** - turns a specified pixel to white.
- **pixelxor** - xor's a specified pixel.
- **pixeltest** - returns the state of a specified pixel.
- **pixeloffhl** - similar to pixeloff.
- **pixelxorhl** - similar to pixelxor.
- **pixeltesthl** - similar to pixeltest.
- **fastlineb** - draws a black line to the graph buffer.
- **fastlinew** - draws a white line to the graph buffer.
- **fastlinex** - XORs a line onto the graph buffer.
- **pointonc** - like pixelon, but with clipping.
- **pointoffc** - like pixeloff, but with clipping.
- **pointxorc** - like pixelxor, but with clipping.
- **centertext** - draws a centered string to the graph buffer in variable-width font.
- **cphlbc** - compares hl and bc.
- **putsprite8** - like iPutSprite width height=8.
- **fastcopyb** - similar to iFastCopy, but using hl instead of gbuf as the memory location.
- **vputsc** - like vputs, but copying to both the LCD and the graph buffer.
- **scroll7** - scroll the graph buffer by 7 lines.

- **vnewline** - like CRLF, calling scroll7 if necessary.
- **rand127** - return a random number between 0 and 127, inclusive.
- **disprle** - decompress RLE-compressed image.
- **cphlde** - compare hl and de.
- **fastlined** - draws a fast line with a dotted line style.
- **disprlel** - similar to disprle, but with a specified size.
- **compstrs** - compare two zero-terminated strings.
- **nextstr** - increment through a zero-terminated string to the following string.
- **fastrectangle** - draws an unfilled rectangle to the graph buffer.
- **gettext** - like vputs, but gets input.
- **gettextv** - same as gettext with additional options.
- **vputa** - displays the value of the accumulator in base 10.
- **compstrsn** - compares two strings for a specified number of characters.
- **largespritehl** - similar to iLargeSprite
- **setupint** - sets up a simplified MirageOS interrupt tasker.

CHAPTER 10

FURTHER READING

The most important place to get information about 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 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 CS is updated regularly to fix any reported bugs and compatibility issues, optimize size, and add new features. You can find all Doors CS news at the Cemeteck homepage, <http://www.Cemeteck.tk>. If you sign up as a Cemeteck user, you can view the project page with beta editions and more at <http://dcs.cemeteck.net>. You can download this and all future editions of Doors CS at the Cemeteck 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 at kerm_martian@yahoo.com with the phrase "Doors CS" in the subject line.

Doors CS is intellectually copyrighted by Christopher Mitchell, programming alias Kerm Martian. "Doors CS", "The Revolutionary New Shell for Graphing Calculators," "Cemeteck," and "Leading the Way to the Future" are copyright ©1998-2010 Christopher Mitchell. Doors CS may not be reverse engineered or modified without express written consent of the author. Doors CS may not be sold or installed for any monetary or other reimbursement. Doors CS may not be repackaged or redistributed without the permission of the author.

The full Doors CS license is reproduced below:

Doors CS 6 End-User License

A.1| Preamble

This license applies to any and all possible pieces of human- and machine-readable computer 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. Any and all use and reuse of this code for a purpose other than an official release of a compiled version by "Kerm Martian" himself is governed by this agreement. Any attempt to use or reuse this 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".

By opening, downloading, or viewing this document or the source code of DOORS CALCULATOR SHELL ("Doors CS"), 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 any source code you have viewed, downloaded, or cached. This license does not covered the compiled Application version of Doors CS, which is released with its own license agreement.

All legal rights accorded copyrighted or protected works not expressly covered in this document are reserved by "Kerm Martian". Note that while Doors CS is partially open-source, the original code remains copyright "Kerm Martian". However, as stated below, portions of

the code may be reused as long as all strictures both implicitly and expressly stated in this license are followed.

A.2| Availability

The Doors CS 6 source code shall be openly available to the general public after May 31, 2006. This availability cannot be applied to past or future Doors CS releases, although the author, "Kerm Martian", may choose to make such source code available at his own discretion. A release date, if any, for the Doors CS 7 source code has not yet been set. Appropriate notifications will be made should such a date be selected. Interested community members may request special access to the source code in whole or in part, which Kerm Martian reserves the right to approve or deny. Past access to any source code or subset thereof does in no way guarantee future access. When and if released, the source code may be found on Cemetechnet, <http://www.cemetechnet.net>, or <http://doorscs.cemetechnet.net>. If the code is not found there, the author may be contacted to obtain the full source code. At any point, "Kerm Martian" may choose to return Doors CS to closed source status, in which state the code would not be released to any programmer, user, or other individual, corporation, or legal body whatsoever. Such a change would be accompanied by an updated license and at least 1-week notification on <http://www.cemetechnet.net>. This change would retroactively forbid derivative works based in whole or part on original code taken from Doors CS; in the event of such a move, an author could submit a written request to retain rights to the derivative code.

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 CS, its derivatives, and its modules. This license does NOT cover any program written to work with Doors CS 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 absolutely no circumstances whatsoever may the source code of Doors CS may be recompiled in whole or in part and released by an individual, group, or third party other than "Kerm Martian" without express, explicit 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 CS 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.

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 CS 7. 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 CS 7, though all reasonable care has been taken to remove instability from this final release.

A.6| Updates

Frequent changes are made to the Doors CS source code as development progresses. These changes are mirrored in the released source code, but there might be a slight delay due to scheduling or "Kerm Martian" forgetting to upload the source on any given week. New source code releases are generally made after major changes or the addition of new features, and do not occur after minor bugfixes. If you need an up-to-date version, contact "Kerm Martian".