



REFERENCE

4D SYSTEMS

TURNING TECHNOLOGY INTO ART

DIABLO16

Internal 4DGL Functions

Document Date: 6th October 2015
Document Revision: 1.15

Contents

1. 4DGL Introduction	11
2. DIABLO16 Chip-Resident Functions Summary	12
2.1. GPIO Functions.....	21
2.1.1. pin_Set(mode, pin).....	22
2.1.2. pin_HI(pin).....	23
2.1.3. pin_LO(pin).....	24
2.1.4. pin_Val(pin).....	25
2.1.5. pin_Read(pin).....	26
2.1.6. bus_Read().....	27
2.1.7. bus_Read8().....	28
2.1.8. bus_Write8(value).....	29
2.1.9. bus_SetPins(value).....	30
2.1.10. bus_ClearPins(value).....	31
2.1.11. bus_SetChangeInterrupt (function, portmask).....	32
2.1.12. Qencoder1(PHApin, PHBpin, mode).....	33
2.1.13. Qencoder1Reset().....	34
2.1.14. Qencoder2(PHApin, PHBpin, mode).....	35
2.1.15. Qencoder2Reset().....	36
2.1.16. pwm_Init(pin, mode, value).....	37
2.1.17. pin_Pulseout(pin, value).....	38
2.1.18. pin_Counter(pin, mode, OVFunction).....	39
2.1.19. ana_HS(rate, samples, IO1buf, IO2buf, IO3buf, IO4buf, userFunction).....	41
2.1.20. pin_PulseoutCount(pin, frequency, count, function).....	42
2.1.21. OW_Reset(pin).....	43
2.1.22. OW_Read(pin).....	44
2.1.23. OW_Read9(pin).....	45
2.1.24. OW_Write(pin, data).....	46
2.1.25. NP_Write(pin, data, size, Options, RepeatFirst, Repeat, RepeatLast).....	48
2.2. System Memory Access Functions.....	50
2.2.1. peekW(address).....	51
2.2.2. pokeW(address, word_value).....	52
2.3. Maths Functions.....	53
2.3.1. ABS(value).....	54
2.3.2. MIN(value1, value2).....	55
2.3.3. MAX(value1, value2).....	56
2.3.4. SWAP(&var1, &var2).....	57
2.3.5. SIN(angle).....	58
2.3.6. COS(angle).....	59
2.3.7. RAND().....	60
2.3.8. RANDVAL(low, high).....	61
2.3.9. SEED(number).....	62
2.3.10. SQRT(number).....	63
2.3.11. OVF().....	64
2.3.12. CY().....	65
2.3.13. EVE_SP().....	66

2.3.14. EVE_SSIZE()	67
2.3.15. uadd_3232(&res32, &val1, &val2)	68
2.3.16. usub_3232(&res32, &val1, &val2)	69
2.3.17. umul_1616(&res32, val1, val2)	70
2.3.18. udiv_3232(&res32, val1, val2)	71
2.3.19. ucmp_3232(&val1, &val2)	72
2.4. Text and String Functions	73
2.4.1. txt_MoveCursor(line, column)	75
2.4.2. putch(char)	76
2.4.3. putchXY(xpos, ypos, char)	77
2.4.4._putstr(pointer)	78
2.4.5._putstrXY(xpos, ypos, string)	80
2.4.6._putstrCentred(xc, yc, string)	81
2.4.7. Putnum(format, value)	82
2.4.8. print(...)	84
2.4.9. to(outstream)	86
2.4.10. charwidth('char')	88
2.4.11. charheight('char')	89
2.4.12. strwidth(pointer)	90
2.4.13. strheight()	91
2.4.14. strlen(pointer)	92
2.4.15. unicode_page(charbeg, charend, charoffset)	93
2.4.16. txt_Set(function, value)	94
2.4.17. txt_FontBank(bank, address)	97
2.4.18. PutnumXY(x, y, format, value)	98
2.5. Ctype Functions	99
2.5.1. isdigit(char)	100
2.5.2. isxdigit(char)	101
2.5.3. isupper(char)	102
2.5.4. islower(char)	103
2.5.5. isalpha(char)	104
2.5.6. isalnum(char)	105
2.5.7. isprint(char)	106
2.5.8. isspace(char)	107
2.5.9. toupper(char)	108
2.5.10. tolower(char)	109
2.5.11. LObyte(var)	110
2.5.12. HIbyte(var)	111
2.5.13. ByteSwap(var)	112
2.5.14. NybleSwap(var)	113
2.6. Graphics Functions	114
2.6.1. gfx_Cls()	116
2.6.2. gfx_ChangeColour(oldColour, newColour)	117
2.6.3. gfx_Circle(x, y, radius, colour)	118
2.6.4. gfx_CircleFilled(x, y, radius, colour)	119
2.6.5. gfx_Line(x1, y1, x2, y2, colour)	120
2.6.6. gfx_Hline(y, x1, x2, colour)	121
2.6.7. gfx_Vline(x, y1, y2, colour)	122
2.6.8. gfx_Rectangle(x1, y1, x2, y2, colour)	123
2.6.9. gfx_RectangleFilled(x1, y1, x2, y2, colour)	124

2.6.10. gfx_RoundRect(x1, y1, x2, y2, rad, oct, colour).....	125
2.6.11. gfx_Polyline(n, vx, vy, colour).....	126
2.6.12. gfx_Polygon(n, vx, vy, colour).....	128
2.6.13. gfx_Triangle(x1, y1, x2, y2, x3, y3, colour)	129
2.6.14. gfx_Dot().....	130
2.6.15. gfx_Bullet(radius)	131
2.6.16. gfx_OrbitInit(&x_dest, &y_dest)	132
2.6.17. gfx_Orbit(angle, distance)	133
2.6.18. gfx_PutPixel(x, y, colour)	134
2.6.19. gfx_GetPixel(x, y).....	135
2.6.20. gfx_MoveTo(xpos, ypos)	136
2.6.21. gfx_MoveRel(xoffset, yoffset)	137
2.6.22. gfx_IncX().....	138
2.6.23. gfx_IncY()	139
2.6.24. gfx_LineTo(xpos, ypos)	140
2.6.25. gfx_LineRel(xpos, ypos)	141
2.6.26. gfx_BoxTo(x2, y2)	142
2.6.27. gfx_SetClipRegion()	143
2.6.28. gfx_Ellipse(x, y, xrad, yrad, colour).....	144
2.6.29. gfx_EllipseFilled(x, y, xrad, yrad, colour)	145
2.6.30. gfx_Button(state, x, y, buttonColour, txtColour, font, txtWidth txtHeight, text).....	146
2.6.31. gfx_Button2(state, x, y, width, height, buttonColour, txtColour, text)	148
2.6.32. gfx_Button3(state, x, y, width, height, buttonColour, txtColour, text)	149
2.6.33. gfx_Panel(state, x, y, width, height, Colour)	150
2.6.34. gfx_RoundPanel(state, x, y, width, height, radius, bevelwidth, Colour)	151
2.6.35. gfx_Slider(mode, x1, y1, x2, y2, colour, scale, value)	152
2.6.36. gfx_Slider2(mode, x1, y1, width, height, colour, scale, value)	153
2.6.37. gfx_ScreenCopyPaste(xs, ys, xd, yd, width, height)	154
2.6.38. gfx_RGBto565(RED, GREEN, BLUE).....	155
2.6.39. gfx_332to565(COLOUR8BIT)	156
2.6.40. gfx_565to332(COLOUR)	157
2.6.41. gfx_TriangleFilled(x1, y1, x2, y2, x3, y3, colour).....	158
2.6.42. gfx_PolygonFilled(n, vx, vy, colour)	159
2.6.43. gfx-Origin(x, y)	160
2.6.44. gfx_Get(mode)	161
2.6.45. gfx_ClipWindow(x1, y1, x2, y2)	162
2.6.46. gfx_Set(function, value)	163
2.6.47. gfx_Arc(xc, yc, radius, step, startangle, endangle, mode).....	166
2.6.48. gfx_CheckBox(state, x, y, Width, Height, boxColour, textColour, text).....	167
2.6.49. gfx_RadioButton(state, x, y, width, height, boxColour, textColour, text)	168
2.6.50. gfx_FillPattern(patptr, mode).....	169
2.6.51. gfx_Gradient(style, x1, y1, x2, y2, color1, color2)	170
2.6.52. gfx_RoundGradient(style, x1, y1, x2, y2, radius, color1, color2).....	171
2.6.53. gfx_PieSlice(cx, cy, spread, radius, step, startangle, endangle, mode)	172
2.6.54. gfx_PointWithinBox(x, y, &rect)	173
2.6.55. gfx_PointWithinRectangle(x, y, &recta)	174
2.6.56. gfx_ReadBresLine(x1, y1, x2, y2, ptr)	175
2.6.57. gfx_WriteBresLine(x1, y1, x2, y2, ptr)	176
2.6.58. gfx_ReadGRAMarea(x1, y1, x2, y2, ptr)	177
2.6.59. gfx_WriteGRAMarea(x1, y1, x2, y2, ptr)	178
2.6.60. gfx_Surround(x1, y1, x2, y2, rad1, rad2, oct, colour)	179

2.6.61. gfx_Scope(Left, Width, Yzero, n, Xstep, Yamp, Colourbg, old_y1, new_y1, Colour1, ... old_y4, new_y4, Colour4)	180
2.7. Display I/O Functions	181
2.7.1. disp_SetReg(register, data)	182
2.7.2. disp_setGRAM(x1, y1, x2, y2).....	183
2.7.3. disp_WrGRAM(colour)	184
2.7.4. disp_WriteControl(value)	185
2.7.5. disp_WriteWord(value)	186
2.7.6. disp_ReadWord(value).....	187
2.7.7. disp_Disconnect()	188
2.7.8. disp_Init()	189
2.7.9. disp_BlitPixelsFromCOM0() or disp_BlitPixelsFromCOM1() or disp_BlitPixelsFromCOM2() or disp_BlitPixelsFromCOM3().....	190
2.8. Media Functions (SD/SDHC Memory Card or Serial Flash chip).....	192
2.8.1. media_Init()	193
2.8.2. media_SetAdd(HIword, LOword)	194
2.8.3. media_SetSector(HIword, LOword)	195
2.8.4. media_RdSector(Destination_Address)	196
2.8.5. media_WrSector(Source_Address)	197
2.8.6. media_ReadByte()	198
2.8.7. media_ReadWord()	199
2.8.8. media_WriteByte(byte_val)	200
2.8.9. media_WriteWord(word_val)	201
2.8.10. media_Flush()	202
2.8.11. media_Image(x, y)	203
2.8.12. media_Video(x, y).....	204
2.8.13. media_VideoFrame(x, y, frameNumber).....	205
2.9. Flash Memory Chip Functions	207
2.9.1. flash_Bank().....	209
2.9.2. flash_Blit1(bank, offset, count, pallete2colour)	210
2.9.3. flash_Blit2(bank, offset, count, pallete4colour).....	211
2.9.4. flash_Blit4(bank, offset, count, pallete16colour)	212
2.9.5. flash_Blit8(bank, offset, count)	213
2.9.6. flash_Blit16(bank, offset, count)	214
2.9.7. flash_Copy(bank, ptr, dest, count)	215
2.9.8. flash_EraseBank(bank, confirmation)	216
2.9.9. flash_Exec(flashbank, arglistptr)	217
2.9.10. flash_GetByte(bank, ptr)	217
2.9.11. flash_GetWord(bank, ptr)	218
2.9.12. flash_LoadFile(bank, filename)	219
2.9.13. flash_putstr(bank, ptr)	221
2.9.14. flash_Run(bank)	222
2.9.15. flash_WriteBlock(sourceptr, bank, page).....	223
2.10. SPI Control Functions	224
2.10.1. spi_Init(speed, input_mode, output_mode)	225
2.10.2. spi_Read().....	226
2.10.3. spi_Write(byte)	227
2.10.4. spi_Disable()	228
2.10.5. SPI1_Init(speed, mode) or SPI2_Init(speed, mode) or SPI3_Init(speed, mode).....	229
2.10.6. SPI1_Read() or SPI2_Read() or SPI3_Read()	231

2.10.7. SPI1_Write(byte) or SPI2_Write(byte) or SPI3_Write(byte)	232
2.10.8. SPI1_SCK_pin(pin) or SPI2_SCK_pin(pin) or SPI3_SCK_pin(pin)	233
2.10.9. SPI1_SDI_pin(pin) or SPI2_SDI_pin(pin) or SPI3_SDI_pin(pin)	234
2.10.10. SPI1_SDO_pin(pin) or SPI2_SDO_pin(pin) or SPI3_SDO_pin(pin)	235
2.10.11. spi_ReadBlock() or spi1_ReadBlock() or spi2_ReadBlock() or spi3_ReadBlock()	236
2.10.12. spi_WriteBlock() or spi1_WriteBlock() or spi2_WriteBlock() or spi3_WriteBlock()	237
2.11. Serial (UART) Communications Functions	237
2.11.1. COM1_RX_pin(pin) or COM2_RX_pin(pin) or COM3_RX_pin(pin)	239
2.11.2. COM1_TX_pin(pin) or COM2_TX_pin(pin) or COM3_TX_pin(pin)	240
2.11.3. setbaud(baudnum)	241
2.11.4. com_SetBaud(comport, baudrate/10)	242
2.11.5. serin() or serin1() or serin2() or serin3()	244
2.11.6. serout(char) or serout1(char) or serout2(char) or serout3(char)	245
2.11.7. com_Init(buffer, bufsize, qualifier) or com1_Init(buffer, bufsize, qualifier) or com2_Init(buffer, bufsize, qualifier) or com3_Init(buffer, bufsize, qualifier)	246
2.11.8. com_Reset() or com1_Reset() or com2_Reset() or com3_Reset()	248
2.11.9. com_Count() or com1_Count() or com2_Count() or com3_Count()	249
2.11.10. com_Full() or com1_Full() or com2_Full() or com3_Full()	250
2.11.11. com_Error() or com1_Error() or com2_Error() or com3_Error()	251
2.11.12. com_Sync() or com1_Sync() or com2_Sync() or com3_Sync()	252
2.11.13. com_TXbuffer(buf, bufsize, pin) or com1_TXbuffer(buf, bufsize, pin) or com2_TXbuffer(buf, bufsize, pin) or com3_TXbuffer(buf, bufsize, pin)	253
2.11.14. com_TXbufferHold(state) or com1_TXbufferHold(state) or com2_TXbufferHold(state) or com3_TXbufferHold(state)	254
2.11.15. com_TXcount() or com1_TXcount() or com2_TXcount() or com3_TXcount()	255
2.11.16. com_TXemptyEvent(function) or com1_TXemptyEvent(function) or com2_TXemptyEvent(function) or com3_TXemptyEvent(function)	256
2.11.17. com_Mode("databits", "parity", "Stopbits", "comport")	259
2.11.18. com_RXblock() or com1_RXblock() or com2_RXblock() or com3_RXblock()	260
2.11.19. com_TXblock() or com1_TXblock() or com2_TXblock() or com3_TXblock()	261
2.12. I2C BUS Master Functions	262
2.12.1. I2C1_Open(Speed, SCL, SDA) or I2C2_Open(Speed, SCL, SDA) or I2C3_Open(Speed, SCL, SDA)	263
2.12.2. I2C1_Close() or I2C2_Close() or I2C3_Close()	264
2.12.3. I2C1_Start() or I2C2_Start() or I2C3_Start()	266
2.12.4. I2C1_Stop() or I2C2_Stop() or I2C3_Stop()	267
2.12.5. I2C1_Restart() or I2C2_Restart() or I2C3_Restart()	268
2.12.6. I2C1_Read() or I2C2_Read() or I2C3_Read()	269
2.12.7. I2C1_Write(byte) or I2C2_Write(byte) or I2C3_Write(byte)	270
2.12.8. I2C1_Ack() or I2C2_Ack() or I2C3_Ack()	271
2.12.9. I2C1_Nack() or I2C2_Nack() or I2C3_Nack()	272
2.12.10. I2C1_AckStatus or I2C2_AckStatus or I2C3_AckStatus	273
2.12.11. I2C1_AckPoll(control) or I2C2_AckPoll(control) or I2C3_AckPoll(control)	274
2.12.12. I2C1_Idle() or I2C2_Idle() or I2C3_Idle()	275
2.12.13. I2C1_Gets(buffer, size) or I2C2_Gets(buffer, size) or I2C3_Gets(buffer, size)	276
2.12.14. I2C1_Getn() or I2C2_Getn() or I2C3_Getn()	277
2.12.15. I2C1_Puts(buffer) or I2C2_Puts(buffer) or I2C3_Puts(buffer)	278
2.12.16. I2C1_Putn() or I2C2_Putn() or I2C3_Putn()	279
2.13. Timer Functions	280
2.13.1. sys_T()	281
2.13.2. sys_T_HI()	282

2.13.3. sys_SetTimer(timernum, value)	283
2.13.4. sys_GetTimer(timernum)	284
2.13.5. sys_SetTimerEvent(timernum, function)	285
2.13.6. sys_EventQueue().....	286
2.13.7. sys_EventsPostpone().....	287
2.13.8. sys_EventsResume()	288
2.13.9. sys_DeepSleep(units)	289
2.13.10. sys_Sleep(units).....	290
2.13.11. iterator(offset).....	291
2.13.12. sys_GetDate()	292
2.13.13. sys_GetTime().....	293
2.13.14. sys_SetDate(year, month, day)	294
2.13.15. sys_SetTime(hour, minute, second).....	295
2.13.16. sys_GetDateVar(&year, &month, &day).....	296
2.13.17. sys_GetTimeVar(&hour, &minute, &second, &msecs)	297
2.14. FAT16 File Functions	298
2.14.1. file_Error()	299
2.14.2. file_Count(filename)	300
2.14.3. file_Dir(filename)	301
2.14.4. file_FindFirst(fname)	302
2.14.5. file_FindNext().....	303
2.14.6. file_Exists(fname)	304
2.14.7. file_Open(fname, mode)	305
2.14.8. file_Close(handle).....	306
2.14.9. file_Read(destination, size, handle)	307
2.14.10. file_Seek(handle, HiWord, LoWord).....	308
2.14.11. file_Index(handle, Hisize, LoSize, recordnum)	309
2.14.12. file_Tell(handle, &HiWord, &LoWord).....	310
2.14.13. file_Write(*source, size, handle).....	311
2.14.14. file_Size(handle, &HiWord, &LoWord)	312
2.14.15. file_Image(x, y, handle).....	313
2.14.16. file_ScreenCapture(x, y, width, height, handle).....	314
2.14.17. file_PutC(char, handle).....	315
2.14.18. file_GetC(handle)	316
2.14.19. file_PutW(word, handle)	317
2.14.20. file_GetW(handle)	318
2.14.21. file_PutS(*source, handle)	319
2.14.22. file_GetS(*string, size, handle).....	320
2.14.23. file_Erase(fname)	321
2.14.24. file_Rewind(handle)	322
2.14.25. file_LoadFunction(fname.4XE)	323
2.14.26. file_Run(fname.4XE, arglistptr)	325
2.14.27. file_Exec(fname.4XE, arglistptr)	330
2.14.28. file_LoadImageControl(fname1, fname2, mode).....	331
2.14.29. file_Mount()	334
2.14.30. file_Unmount().....	336
2.14.31. file_PlayWAV(fname)	337
2.14.32. file_Rename(oldname, newname).....	338
2.14.33. file_SetDate(handle, year, month, day, hour, minute, second).....	339
2.15. Sound Control Functions.....	340

2.15.1. Snd_Volume(var).....	341
2.15.2. Snd_Pitch(pitch)	342
2.15.3. Snd_BufSize(var).....	343
2.15.4. snd_Stop()	344
2.15.5. snd_Pause()	345
2.15.6. snd_Continue()	346
2.15.7. snd_Playing()	347
2.15.8. snd_Freq(frequency, duration)	348
2.16. String Class Functions.....	349
2.16.1. str_Ptr(&var).....	350
2.16.2. str_GetD(&ptr, &var).....	351
2.16.3. str_GetW(&ptr, &var).....	352
2.16.4. str_GetHexW(&ptr, &var)	353
2.16.5. str_GetC(&ptr, &var)	354
2.16.6. str_GetByte(ptr)	355
2.16.7. str_GetWord(ptr)	356
2.16.8. str_PutByte(ptr, val)	357
2.16.9. str_PutWord(ptr, val)	358
2.16.10. str_Match(&ptr, *str)	359
2.16.11. str_MatchI(&ptr, *str)	360
2.16.12. str_Find(&ptr, *str).....	361
2.16.13. str_FindI(&ptr, *str).....	362
2.16.14. str_Length(ptr)	363
2.16.15. str_Printf(&ptr, *format).....	364
2.16.16. str_Cat(&destination, &source)	366
2.16.17. str_CatN(&ptr, str, count)	367
2.16.18. str_ByteMove(src, dest, count)	368
2.16.19. str_Copy(dest, src).....	369
2.16.20. str_CopyN(dest, src, count).....	370
2.17. Touch Screen Functions	371
2.17.1. touch_DetectRegion(x1, y1, x2, y2)	372
2.17.2. touch_Set(mode).....	373
2.17.3. touch_Get(mode)	374
2.17.4. touch_TestArea(&rect).....	375
2.17.5. touch_TestBox(&rect)	376
2.18. Image Control Functions	377
2.18.1. img_SetPosition(handle, index, xpos, ypos)	378
2.18.2. img_Enable(handle, index)	379
2.18.3. img_Disable(handle, index)	380
2.18.4. img_Darken(handle, index)	381
2.18.5. img_Lighten(handle, index).....	382
2.18.6. img_SetWord(handle, index, offset, word)	383
2.18.7. img_GetWord(handle, index, offset).....	384
2.18.8. img_Show(handle, index)	385
2.18.9. img_SetAttributes(handle, index, value)	386
2.18.10. img_ClearAttributes(handle, index, value).....	387
2.18.11. img_Touched(handle, index).....	388
2.18.12. img_SelectReadPosition(handle, index, frame, xpos, ypos).....	389
2.18.13. img_SequentialRead(count, ptr)	390
2.19. Memory Allocation Functions	391

2.19.1. mem_Alloc(size)	392
2.19.2. mem_AllocV(size)	393
2.19.3. mem_Allocz(size).....	394
2.19.4. mem_Realloc(&ptr, size)	395
2.19.5. mem_Free(allocation)	396
2.19.6. mem_Heap().....	397
2.19.7. mem_Set(ptr, char, size)	398
2.19.8. mem_Copy(source, destination, count).....	399
2.19.9. mem_Compare(ptr1, ptr2, count).....	400
2.19.10. mem_ArrayOp1(memarray, count, op, value)	401
2.19.11. mem_ArrayOp2(memarray1, memarray2, count, op, value)	403
2.20. General Purpose Functions	404
2.20.1. pause(time)	405
2.20.2. lookup8(key, byteConstList)	406
2.20.3. lookup16(key, wordConstList).....	407
2.21. Floating point Functions.....	408
2.21.1.flt_ADD(&result, &floatA, &floatB)	409
2.21.2.flt_SUB(&result, &floatA, &floatB).....	410
2.21.3.flt_MUL(&result, &floatA, &floatB).....	411
2.21.4.flt_DIV(&result, &floatA, &floatB).....	412
2.21.5.flt_POW(&result, &floatA, &floatB)	413
2.21.6.flt_ABS(&result, &floatval)	414
2.21.7.flt_CEIL(&result, &floatval).....	415
2.21.8.flt_FLOOR(&result, &floatval)	416
2.21.9.flt_SIN(&result, &floatval)	417
2.21.10.flt_COS(&result, &floatval).....	418
2.21.11.flt_TAN(&result, &floatval)	419
2.21.12.flt_ASIN(&result, &floatval).....	420
2.21.13.flt_ACOS(&result, &floatval)	421
2.21.14.flt_ATAN(&result, &floatval)	422
2.21.15.flt_EXP(&result, &floatval)	423
2.21.16.flt_LOG(&result, &floatval)	424
2.21.17.flt_SQR(&result, &floatval).....	425
2.21.18.flt_LT(&floatA, &floatB).....	426
2.21.19.flt_EQ(&floatA, &floatB).....	427
2.21.20.flt_NE(&floatA, &floatB).....	428
2.21.21.flt_GT(&floatA, &floatB).....	429
2.21.22.flt_GE(&floatA, &floatB).....	430
2.21.23.flt_LE(&floatA, &floatB).....	431
2.21.24.flt_SGN(&floatval)	432
2.21.25.flt_FTOI(&floatval).....	433
2.21.26.flt_ITOF(&fresult, var16)	434
2.21.27.flt_UITOF(&fresult, uvar16).....	435
2.21.28.flt_LTOF(&fresult, var32).....	436
2.21.29.flt_ULTOF(&fresult, uvar32)	437
2.21.30.flt_VAL(&fresult, numstring)	438
2.21.31.flt_PRINT (&fvalue, formatstring).....	439
2.21.32.flt_PRINTxy (x, y, &fvalue, formatstring).....	440
2.22. Misc System Functions	441
2.22.1. sys_PmmC()	442

2.22.2. sys_Driver().....	443
2.23. SPI FLASH Functions	444
2.23.1. spiflash_BlockErase(spi#, Enablepin, block)	445
2.23.2. spiflash_BulkErase(spi#, Enablepin)	446
2.23.3. spiflash_Exec(spi#, Enablepin, arglistptr)	447
2.23.4. spiflash_GetC(spi#, Enablepin)	448
2.23.5. spiflash_GetS(*String, size, spi#, Enablepin)	449
2.23.6. spiflash_GetW(spi#, Enablepin)	450
2.23.7. spiflash_ID(spi#, Enablepin)	451
2.23.8. spiflash_Image(x, y, spi#, Enablepin).....	452
2.23.9. spiflash_LoadFunction(spi#, Enablepin)	453
2.23.10. spiflash_LoadImageControl(spi#, Enablepin)	455
2.23.11. spiflash_PlayWAV(spi#, Enablepin)	458
2.23.12. spiflash_PutC(char, spi#, Enablepin)	459
2.23.13. spiflash_PutS(source, spi#, Enablepin).....	460
2.23.14. spiflash_PutW(word, spi#, Enablepin)	461
2.23.15. spiflash_Read(destination, size, spi#, Enablepin).....	462
2.23.16. spiflash_Run(spi#, Enablepin, arglistptr)	463
2.23.17. spiflash_SetAdd(spi#, HiWord, LoWord).....	464
2.23.18. spiflash_SIG(spi#, Enablepin)	465
2.23.19. spiflash_Write(Source, size, spi#, Enablepin).....	466
2.24. CRC Functions.....	467
2.24.1. crc_16(buf, count)	468
2.24.2. crc_CCITT(buf, count, seed)	469
2.24.3. crc_CSUM_8(buf, count)	470
2.24.4. crc_MODBUS(buf, count)	471
3. System Registers Memory Map	472
4. Appendix A : Runtime Error Messages	474
5. Hardware Tools	475
5.1. 4D Programming Cable & Adaptor	475
5.2. Evaluation Display Modules	475
5.3. Memory Cards - FAT16 Format	475
6. 4D Systems - Workshop 4 IDE.....	476
6.1. Workshop 4 – Designer Environment	477
6.2. Workshop 4 – ViSi Environment.....	477
6.3. Workshop 4 – ViSi Genie Environment	477
6.4. Workshop 4 – Serial Environment.....	478
7. Revision History.....	479
8. Legal Notice.....	480
9. Contact Information	480

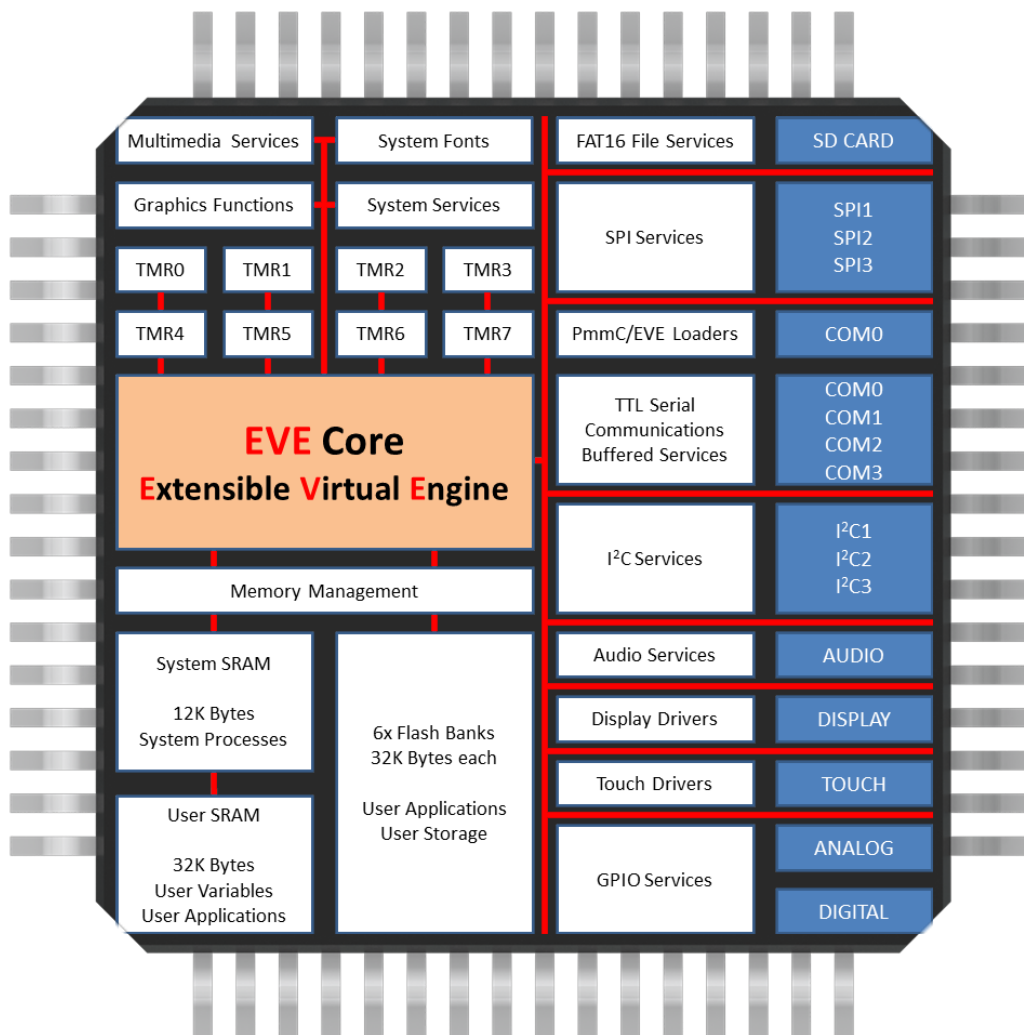
1. 4DGL Introduction

The 4D-Labs family of embedded graphics processors (GOLDELOX, PICASO and DIABLO16) are powered by a highly optimised soft core virtual engine, E.V.E. (Extensible Virtual Engine). EVE was designed and created by 4D Systems in the early 2000's, and should not be confused by FTDI's solution of EVE, which was developed a decent decade or so later.

EVE is a proprietary, high performance virtual processor with an extensive byte-code instruction set optimised to execute compiled 4DGL programs. 4DGL (4D Graphics Language) was specifically developed from ground up for the EVE engine core. It is a high level language which is easy to learn and simple to understand yet powerful enough to tackle many embedded graphics applications.

4DGL is a graphics oriented language allowing rapid application development. An extensive library of graphics, text and file system functions and the ease of use of a language that combines the best elements and syntax structure of languages such as C, Basic, Pascal, etc. Programmers familiar with these languages will feel right at home with 4DGL. It includes many familiar instructions such as IF..ELSE..ENDIF, WHILE..WEND, REPEAT..UNTIL, GOSUB..ENDSUB, GOTO as well as a wealth of (chip-resident) internal functions that include SERIN, SEROUT, GFX_LINE, GFX_CIRCLE and many more.

This document covers the internal (chip-resident) functions available for the DIABLO16 Processor. This document should be used in conjunction with **"4DGL-Programmers-Reference-Manual"** document.



DIABLO16 Internal Block Diagram

2. DIABLO16 Chip-Resident Functions Summary

The following is a summary of chip-resident 4DGL functions within the DIABLO16 graphics processor. The document is made up of the following sections:

2.1 GPIO Functions:

- pin_Set(mode, pin)
- pin_HI(pin)
- pin_LO(pin)
- pin_Val(pin, value)
- pin_Read(pin)
- bus_Read()
- bus_Read8()
- bus_Write8(value)
- bus_SetPins(value)
- bus_ClearPins(value)
- bus_SetChangeInterrupt(function, portmask)
- Qencoder1(PHApin, PHBpin, mode)
- Qencoder1Reset()
- Qencoder2(PHApin, PHBpin, mode)
- Qencoder2Reset()
- PWM_Init(pin, mode, value)
- pin_Pulseout(pin, value) or pin_PulseoutB(pin, value)
- pin_Counter(pin, mode, OVfunction)
- ana_HS(rate, samples, IO1buf, IO2buf, IO3buf, IO4buf, userFunction)
- OW_Reset(pin)
- OW_Read(pin)
- OW_Read9(pin)
- OW_Write(pin, data)
- NP_Write(pin, data, size, Options, RepeatFirst, Repeat, RepeatLast)

2.2 System Memory Access Functions:

- peekW(address)
- pokeW(address, wordvalue)

2.3 Maths Functions:

- ABS(value)
- MIN(value1, value2)
- MAX(value1, value2)
- SWAP(&var1, &var2)
- SIN(angle)
- COS(angle)
- RAND()
- RANDVAL(low, high)
- SEED(number)
- SQRT(number)
- OVF ()
- CY()
- EVE_SP()
- EVE_SSIZE()
- umul_1616(&res32, val1, val2)
- uadd_3232(&res32, &val1, &val2)
- usub_3232(&res32, &val1, &val2)
- udiv_3232(&res32, &var1, &var2)
- ucmp_3232(&val1, &val2)

2.4 Text and String Functions:

- txt_MoveCursor(line, column)

- `putch(char)`
- `putchXY(xpos, ypos, char)`
- `putstr(pointer)`
- `putstrXY(xpos, ypos, string)`
- `putstrCentred(xc, yc, string)`
- `putnum(format, value)`
- `print(...)`
- `to(outstream)`
- `charwidth(char)`
- `charheight(char)`
- `strwidth(pointer)`
- `strheight()`
- `strlen(pointer)`
- `unicode_page(charbeg, charend, charoffset)`
- `txt_Set(function, value)`

txt_Set shortcuts:

- `txt_FGcolour(colour)`
- `txt_BGcolour(colour)`
- `txt_FontID(id)`
- `txt_Width(multiplier)`
- `txt_Height(multiplier)`
- `txt_Xgap(pixelcount)`
- `txt_Ygap(pixelcount)`
- `txt_Delay(millisecs) [deprecated]`
- `txt_Opacity(mode)`
- `txt_Bold(mode)`
- `txt_Italic(mode)`
- `txt_Inverse(mode)`
- `txt_Underline(mode)`
- `txt_Attributes(value)`
- `txt_Wrap(value)`
- `txt_Angle(value)`
- `txt_FontBank(bank, address)`
- `PutnumXY(x, y, format, value)`

2.5 CType Functions:

- `isdigit(char)`
- `isxdigit(char)`
- `isupper(char)`
- `islower(char)`
- `isalpha(char)`
- `isalnum(char)`
- `isprint(char)`
- `isspace(char)`
- `iswhite(char)`
- `toupper(char)`
- `tolower(char)`
- `LByte(var)`
- `HByte(var)`
- `ByteSwap(var)`

2.6 Graphics Functions:

- `gfx_Cls()`
- `gfx_ChangeColour(oldColour, newColour)`
- `gfx_Circle(x, y, radius, colour)`
- `gfx_CircleFilled(x, y, radius, colour)`
- `gfx_Line(x1, y1, x2, y2, colour)`

- gfx_Hline(y, x1, x2, colour)
- gfx_Vline(x, y1, y2, colour)
- gfx_Rectangle(x1, y1, x2, y2, colour)
- gfx_RectangleFilled(x1, y1, x2, y2, colour)
- gfx_RoundRect(x1, y1, x2, y2, rad, colour)
- gfx_Polyline(n, vx, vy, colour)
- gfx_Polygon(n, vx, vy, colour)
- gfx_Triangle(x1, y1, x2, y2, x3, y3, colour)
- gfx_Dot()
- gfx_Bullet(radius)
- gfx_OrbitInit(&x_dest, &y_dest)
- gfx_Orbit(angle, distance)
- gfx_PutPixel(x, y, colour)
- gfx_GetPixel(x, y)
- gfx_MoveTo(xpos, ypos)
- gfx_MoveRel(xoffset, yoffset)
- gfx_IncX()
- gfx_IncY()
- gfx_LineTo(xpos, ypos)
- gfx_LineRel(xpos, ypos)
- gfx_BoxTo(x2, y2)
- gfx_SetClipRegion()
- gfx_Ellipse(x, y, xrad, yrad, colour)
- gfx_EllipseFilled(x, y, xrad, yrad, colour)
- gfx_Button(state, x, y, buttonColour, textColour, font, textWidth, textHeight, text)
- gfx_Button2(state, x, y, width, height, buttonColour, txtColour, text)
- gfx_Button3(state, x, y, width, height, buttonColour, txtColour, text)
- gfx_Panel(state, x, y, width, height, colour)
- gfx_RoundPanel(states, x, y, width, height, radius, bevelwidth, colour)
- gfx_Slider(mode, x1, y1, x2, y2, colour, scale, value)
- gfx_Slider2(mode, x1, y1, width, height, colour, scale, value)
- gfx_ScreenCopyPaste(xs, ys, xd, yd, width, height)
- gfx_RGBto565(RED, GREEN, BLUE)
- gfx_332to565(COLOUR8BIT)
- gfx_565to332(COLOUR)
- gfx_TriangleFilled(x1, y1, x2, y2, x3, y3, colr)
- gfx_PolygonFilled(n, &vx, &vy, colr)
- gfx-Origin(x, y)
- gfx_Get(mode)
- gfx_ClipWindow(x1, y1, x2, y2)
- gfx_Set(function, value)
- **gfx_Set shortcuts:**
 - gfx_PenSize(mode)
 - gfx_BGcolour(colour)
 - gfx_ObjectColour(colour)
 - gfx_Clipping(mode)
 - gfx_TransparentColour(colour)
 - gfx_Transparency(mode)
 - gfx_FrameDelay(delay)
 - gfx_ScreenMode(orientation)
 - gfx_OutlineColour(colour)
 - gfx_Contrast(value)
 - gfx_LinePattern(pattern)
 - gfx_BevelRadius (radius)
 - gfx_BevelWidth(mode)
 - gfx_BevelShadow(value)
 - gfx_Xorigin(offset)

- gfx_Yorigin(offset)
- gfx_Arc(xc, radius, step, startangle, endangle, mode)
- gfx_CheckBox(state, x, y, width, height, boxColour, textColour, text)
- gfx_RadioButton(state, x, y, width, height, boxColour, textColour, text)
- gfx_FillPattern(patptr, mode)
- gfx_Gradient(style, x1, y1, x2, y2, colour1, colour2)
- gfx_RoundGradient(style, x1, y1, x2, y2, radius, colour1, colour2)
- gfx_PieSlice(cx, cy, spread, radius, step, startangle, endangle, mode)
- gfx_PointWithinBox(x, y, &rect)
- gfx_PointWithinRectangle(x, y, &recta)
- gfx_ReadBresLine(x1, y1, x2, y2, ptr)
- gfx_WriteBresLine(x1, y1, x2, y2, ptr)
- gfx_ReadGRAMarea(x1, y1, x1, y2, ptr)
- gfx_WriteGRAMarea(x1, y1, x2, y2, ptr)
- gfx_Surround(x1, y1, x2, y2, rad1, rad2, oct, colour)
- gfx_Scope(Left, Width, Yzero, n, Xstep, Yamp, Colourbg, &old_y1, &new_y1, Colour1, ... &old_y4, &new_y4, Colour4)

2.7 Display I/O Functions:

- disp_SetReg(register, data)
- disp_setGRAM(x1, y1, x2, y2)
- disp_WrGRAM(colour)
- disp_WriteControl(value)
- disp_WriteWord(value)
- disp_ReadWord()
- disp_Sync(line)
- disp_Disconnect()
- disp_Init()

2.8 Media Functions (SD/SDHC memory Card or Serial Flash chip):

- media_Init()
- media_SetAdd(HIword, LOword)
- media_SetSector(HIword, LOword)
- media_RdSector(Destination_Address)
- media_WrSector(Source_Address)
- media_ReadByte()
- media_ReadWord()
- media_WriteByte(byte_val)
- media_WriteWord(word_val)
- media_Flush()
- media_Image(x, y)
- media_Video(x, y)
- media_VideoFrame(x, y, frameNumber)

2.9 Flash Memory chip Functions:

- flash_Bank()
- flash_Blit1(bank, offset, count, pallete2colour)
- flash_Blit16(bank, offset, count)
- flash_Blit2(bank, offset, count, pallete4colour)
- flash_Blit4(bank, offset, count, pallete16colour)
- flash_Blit8(bank, offset, count)
- flash_Copy(bank, ptr, dest, count)
- flash_EraseBank(bank, confirmation)
- flash_Exec(bank, arglistptr)
- flash_GetByte(bank, ptr)
- flash_GetWord(bank, ptr)
- flash_LoadFile(bank, filename)
- flash_putstr(bank, ptr)

- flash_Run(bank)
- flash_WriteBlock(sourceptr, bank, page)

2.10 SPI Control Functions:

- spi_Init(speed, input_mode, output_mode)
- spi_Read()
- spi_Write(byte)
- spi_Disable()
- SPI1_Init(speed, mode) **or** SPI2_Init(speed, mode) **or** SPI3_Init(speed, mode)
- SPI1_Read() **or** SPI2_Read() **or** SPI3_Read()
- SPI1_Write(byte) **or** SPI2_Write(byte) **or** SPI3_Write(byte)
- SPI1_SCK_pin(pin) **or** SPI2_SCK_pin(pin) **or** SPI3_SCK_pin(pin)
- SPI1_SDI_pin(pin) **or** SPI2_SDI_pin(pin) **or** SPI3_SDI_pin(pin)
- SPI1_SDO_pin(pin) **or** SPI2_SDO_pin(pin) **or** SPI3_SDO_pin(pin)

2.11 Serial (UART) Communications Functions:

- COM1_RX_pin(pin) **or** COM2_RX_pin(pin) **or** COM3_RX_pin(pin)
- COM1_TX_pin(pin) **or** COM2_TX_pin(pin) **or** COM3_TX_pin(pin)
- setbaud(rate)
- com_SetBaud(comport, baudrate/10)
- serin() **or** serin1() **or** serin2() **or** serin3()
- serout(char) **or** serout1(char) **or** serout2(char) **or** serout3(char)
- com_Init(buffer, buffsize, qualifier) **or** com_Init1(buffer, buffsize, qualifier) **or** com_Init2(buffer, buffsize, qualifier) **or** com_Init3(buffer, buffsize, qualifier)
- com_Reset() **or** com1_Reset() **or** com2_Reset() **or** com3_Reset()
- com_Count() **or** com1_Count() **or** com2_Count() **or** com3_Count()
- com_Full() **or** com1_Full() **or** com2_Full() **or** com3_Full()
- com_Error() **or** com1_Error() **or** com2_Error() **or** com3_Error()
- com_Sync() **or** com1_Sync() **or** com2_Sync() **or** com3_Sync()
- com_TXbuffer(buf, buffsize, pin) **or** com1_TXbuffer(buf, buffsize, pin) **or** com2_TXbuffer(buf, buffsize, pin) **or** com3_TXbuffer(buf, buffsize, pin)
- com_TXbufferHold(state) **or** com1_TXbufferHold(state) **or** com2_TXbufferHold(state) **or** com3_TXbufferHold(state)
- com_TXcount() **or** com1_TXcount() **or** com2_TXcount() **or** com3_TXcount()
- com_TXemptyEvent(function) **or** com1_TXemptyEvent(function) **or** com2_TXemptyEvent(function) **or** com3_TXemptyEvent(function)

2.12 I2C BUS Master Function

- I2C1_Open(Speed, SCLpin, SDApin) **or** I2C2_Open(Speed, SCLpin, SDApin) **or** I2C3_Open(Speed, SCLpin, SDApin)
- I2C1_Close() **or** I2C2_Close() **or** I2C3_Close()
- I2C1_Start() **or** I2C2_Start() **or** I2C3_Start()
- I2C1_Stop() **or** I2C2_Stop() **or** I2C3_Stop()
- I2C1_Restart() **or** I2C2_Restart() **or** I2C3_Restart()
- I2C1_Read() **or** I2C2_Read() **or** I2C3_Read()
- I2C1_Write(byte) **or** I2C2_Write(byte) **or** I2C3_Write(byte)
- I2C1_Ack() **or** I2C2_Ack() **or** I2C3_Ack()
- I2C1_Nack() **or** I2C2_Nack() **or** I2C3_Nack()
- I2C1_AckStatus() **or** I2C2_AckStatus() **or** I2C3_AckStatus()
- I2C1_AckPoll(control) **or** I2C2_AckPoll(control) **or** I2C3_AckPoll(control)
- I2C1_Idle() **or** I2C2_Idle() **or** I2C3_Idle()
- I2C1_Gets(buffer, size) **or** I2C2_Gets(buffer, size) **or** I2C3_Gets(buffer, size)
- I2C1_Getn(buffer, size) **or** I2C2_Getn(buffer, size) **or** I2C3_Getn(buffer, size)
- I2C1_Puts(buffer) **or** I2C2_Puts(buffer) **or** I2C3_Puts(buffer)
- I2C1_Putn(buffer, count) **or** I2C2_Putn(buffer, count) **or** I2C3_Putn(buffer, count)

2.13 Timer Functions:

- sys_T()

- sys_T_HI()
- sys_SetTimer(timernum, value)
- sys_GetTimer(timernum)
- sys_SetTimerEvent("timernum","function")
- sys_EventQueue()
- sys_EventsPostpone()
- sys_EventsResume()
- sys_DeepSleep(units)
- sys_Sleep(units)
- iterator(offset)
- sys_GetDate()
- sys_GetTime()
- sys_SetDate(year, month, day)
- sys_SetTime(hours, mins, secs)
- sys_GetDateVar(&year, &month, &day)
- sys_GetTimeVar(&hour, &minute, &second, &msecs)

2.14 FAT16 File Functions:

- file_Error()
- file_Count(filename)
- file_Dir(filename)
- file_FindFirst(fname)
- file_FindNext()
- file_Exists(fname)
- file_Open(fname, mode)
- file_Close(handle)
- file_Read(destination, size, handle)
- file_Seek(handle, HiWord, LoWord)
- file_Index(handle, Hisize, Losize, recordnum)
- file_Tell(handle, &HiWord, &LoWord)
- file_Write(Source, size, handle)
- file_Size(handle, &HiWord, &LoWord)
- file_Image(x, y, handle)
- file_ScreenCapture(x, y, width, height, handle)
- file_PutC(char, handle)
- file_GetC(handle)
- file_PutW(word, handle)
- file_GetW(handle)
- file_PutS(source, handle)
- file_GetS(*String, size, handle)
- file_Erase(fname)
- file_Rewind(handle)
- file_LoadFunction(fname.4XE)
- file_Run(fname..4XE, arglistptr)
- file_Exec(fname..4XE, arglistptr)
- file_LoadImageControl(fname1, fname2, mode)
- file_Mount()
- file_Unmount()
- file_PlayWAV
- file_Rename(oldname, newname)
- file_SetDate(handle, year, month, day, hour, minute, second)

2.15 Sound Control Functions:

- snd_Volume(var)
- snd_Pitch(pitch)
- snd_BufSize(var)
- snd_Stop()

- `snd_Pause()`
- `snd_Continue()`
- `snd_Playing()`
- `snd_Freq()`

2.16 String Class Functions:

- `str_Ptr(&var)`
- `str_GetD(&ptr, &var)`
- `str_GetW(&ptr, &var)`
- `str_GetHexW(&ptr, &var)`
- `str_GetC(&ptr, &var)`
- `str_GetByte(ptr)`
- `str_GetWord(ptr)`
- `str_PutByte(ptr, val)`
- `str_PutWord(ptr, val)`
- `str_Match(&ptr, *str)`
- `str_MatchI(&ptr, *str)`
- `str_Find(&ptr, *str)`
- `str_FindI(&ptr, *str)`
- `str_Length(ptr)`
- `str_Printf(&ptr, *format)`
- `str_Cat(&destination, &Source)`
- `str_CatN(&ptr, str, count)`
- `str_ByteMove(src, dest, count)`
- `str_Copy(dest, src)`
- `str_CopyN(dest, src, count)`

2.17 Touch Screen Functions:

- `touch_DetectRegion(x1, y1, x2, y2)`
- `touch_Set(mode)`
- `touch_Get(mode)`
- `touch_TestArea(&rect)`
- `touch_TestBox(&rect)`

2.18 Image Control Functions:

- `img_SetPosition(handle, index, xpos, ypos)`
- `img_Enable(handle, index)`
- `img_Disable(handle, index)`
- `img_Darken(handle, index)`
- `img_Lighten(handle, index)`
- `img_SetWord(handle, index, offset, word)`
- `img_GetWord(handle, index, offset)`
- `img_Show(handle, index)`
- `img_SetAttributes(handle, index, value)`
- `img_ClearAttributes(handle, index, value)`
- `img_Touched(handle, index)`
- `img_SelectReadPosition(handle, index, frame, x, y)`
- `img_SequentialRead(count, ptr)`

2.19 Memory Allocation Functions:

- `mem_Alloc(size)`
- `mem_Allocv(size)`
- `mem_Allocz(size)`
- `mem_Realloc(ptr, size)`
- `mem_Free(allocation)`
- `mem_Heap()`
- `mem_Set(ptr, char, size)`
- `mem_Copy(source, destination, count)`

- mem_Compare(ptr1, ptr2, count)
- mem_ArrayOp1(memarray, count, op, value)
- mem_ArrayOP2(memarray1, memarray2, count, op, value)

2.20 General Purpose Functions:

- pause(milliseconds)
- lookup8 (**key**, byteConstList)
- lookup16 (**key**, wordConstList)

2.21 Floating Point Functions:

- flt_ADD(&result, &floatA, &floatB)
- flt_SUB(&result, &floatA, &floatB)
- flt_MUL(&result, &floatA, &floatB)
- flt_DIV(&result, &floatA, &floatB)
- flt_POW(&result, &floatA, &floatB)
- flt_ABS(&result, &floatval)
- flt_CEIL(&result, &floatval)
- flt_FLOOR(&result, &floatval)
- flt_SIN(&result, &floatval)
- flt_COS(&result, &floatval)
- flt_TAN(&result, &floatval)
- flt_ASIN(&result, &floatval)
- flt_ACOS(&result, &floatval)
- flt_ATN(&result, &floatval)
- flt_EXP(&result, &floatval)
- flt_LOG(&result, &floatval)
- flt_SQR(&result, &floatval)
- flt_LT(&floatA, &floatB)
- flt_EQ(&floatA, &floatB)
- flt_NE(&floatA, &floatB)
- flt_GT(&floatA, &floatB)
- flt_GE(&floatA, &floatB)
- flt_LE(&floatA, &floatB)
- flt_SGN(&floatval)
- flt_FTOI(&floatval)
- flt_ITOF(&fresult, &var16)
- flt_UITOF(&fresult, &uvar16)
- flt_LTOF(&fresult, &var32)
- flt_ULTOF(&fresult, &uvar32)
- flt_VAL(&float1, mystring)
- flt_PRINT(&fvalue, formatstring)
- flt_PRINTxy(x, y, &fvalue, formatstring)

2.22 Misc System Functions:

- sys_PmmC()
- sys_Driver()

2.23 SPI Flash Functions:

- spiflash_BlockErase(spi#, Enablepin, block)
- spiflash_BulkErase(spi#, Enablepin)
- spiflash_Exec(spi#, Enablepin, arglistptr)
- spiflash_GetC(spi#, Enablepin)
- spiflash_GetS(*String, size, spi#, Enablepin)
- spiflash_GetW(spi#, Enablepin)
- spiflash_ID(spi#, Enablepin)
- spiflash_Image(x, y, spi#, Enablepin)
- spiflash_LoadFunction(spi#, Enablepin)

- spiflash_LoadImageControl(spi#, Enablepin)
- spiflash_PlayWAV(spi#, Enablepin)
- spiflash_PutC(char, spi#, Enablepin)
- spiflash_PutS(source, spi#, Enablepin)
- spiflash_PutW(word, spi#, Enablepin)
- spiflash_Read(destination, size, spi#, Enablepin)
- spiflash_Run(spi#, Enablepin, arglistptr)
- spiflash_SetAdd(spi#, HiWord, LoWord)
- spiflash_SIG(spi#, Enablepin)
- spiflash_Write(Source, size, spi#, Enablepin)

2.24 CRC Functions:

- crc_16(buf, count)
- crc_CCITT(buf, count, seed)
- crc_CSUM_8(buf, count)
- crc_MODBUS(buf, count)

2.1. GPIO Functions

Summary of Functions in this section:

- pin_Set(mode, pin)
- pin_HI(pin)
- pin_LO(pin)
- pin_Val(pin, value)
- pin_Read(pin)
- bus_Read()
- bus_Read8()
- bus_Write(value)
- bus_SetPins(value)
- bus_ClearPins(value)
- bus_SetChangeInterrupt(function, portmask)
- Qencoder1(PHApin, PHBpin, mode)
- Qencoder1Reset()
- Qencoder2(PHApin, PHBpin, mode)
- Qencoder2Reset()
- PWM_Init(pin, mode, value)
- pin_Pulseout(pin, value) **or** pin_PulseoutB(pin, value)
- pin_Counter(pin, mode, OVFunction)
- ana_HS(rate, samples, IO1buf, IO2buf, IO3buf, IO4buf, userFunction)
- OW_Reset(pin)
- OW_Read(pin)
- OW_Read9(pin)
- OW_Write(pin, data)
- NP_Write(pin, data, size, Options, RepeatFirst, Repeat, RepeatLast)

2.1.1. pin_Set(mode, pin)

Syntax	<code>pin_Set(mode, pin);</code>																																																																																																																																																																																																							
Arguments	mode, pin																																																																																																																																																																																																							
	mode	A value (usually a constant) specifying the pin operation.																																																																																																																																																																																																						
	pin	A value (usually a constant) specifying the pin number.																																																																																																																																																																																																						
	The arguments can be a variable, array element, expression or constant.																																																																																																																																																																																																							
Returns	nothing																																																																																																																																																																																																							
Description	<p>There are pre-defined constants for mode and pin:</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th></th> <th>PA0</th> <th>PA1</th> <th>PA2</th> <th>PA3</th> <th>PA4</th> <th>PA5</th> <th>PA6</th> <th>PA7</th> <th>PA8</th> <th>PA9</th> <th>PA10</th> <th>PA11</th> <th>PA12</th> <th>PA13</th> <th>PA14</th> <th>PA15</th> </tr> <tr> <th>DIABLO16 Pin Number</th> <th></th> <td>pin 61</td> <td>pin 62</td> <td>pin 63</td> <td>pin 64</td> <td>pin 46</td> <td>pin 49</td> <td>pin 50</td> <td>pin 51</td> <td>pin 52</td> <td>pin 53</td> <td>pin 43</td> <td>pin 44</td> <td>pin 31</td> <td>pin 32</td> <td>pin 37</td> <td>pin 36</td> </tr> <tr> <th>H1 Pin Number</th> <th></th> <td>pin 1</td> <td>pin 3</td> <td>pin 5</td> <td>pin 7</td> <td>pin 29</td> <td>pin 27</td> <td>pin 25</td> <td>pin 23</td> <td>pin 21</td> <td>pin 19</td> <td>pin 8</td> <td>pin 6</td> <td>pin 28</td> <td>pin 30</td> <td>pin 24</td> <td>pin 26</td> </tr> <tr> <th>Pin Mode (Predefined)</th> <th>mode #</th> <th colspan="16">Generic PIN I/O Legal Settings</th> </tr> </thead> <tbody> <tr> <td>PIN_INP</td> <td>0</td> <td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td> </tr> <tr> <td>PIN_INP_HI</td> <td>1</td> <td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td> </tr> <tr> <td>PIN_INP_LO</td> <td>2</td> <td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td> </tr> <tr> <td>PIN_OUT</td> <td>3</td> <td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td> </tr> <tr> <td>PIN_OUT_OD</td> <td>4</td> <td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td> </tr> <tr> <td>PIN_AN</td> <td>5</td> <td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td> </tr> <tr> <td>PIN_ANAVG</td> <td>6</td> <td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td> </tr> </tbody> </table> <p>Note: If using PIN_AN or PIN_ANAVG via the pin_Read() function, then if Touch is enabled this function should be called no more than once per millisecond, otherwise touch behaviour could be erratic.</p>		4D Pin Name (Predefined)		PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7	PA8	PA9	PA10	PA11	PA12	PA13	PA14	PA15	DIABLO16 Pin Number		pin 61	pin 62	pin 63	pin 64	pin 46	pin 49	pin 50	pin 51	pin 52	pin 53	pin 43	pin 44	pin 31	pin 32	pin 37	pin 36	H1 Pin Number		pin 1	pin 3	pin 5	pin 7	pin 29	pin 27	pin 25	pin 23	pin 21	pin 19	pin 8	pin 6	pin 28	pin 30	pin 24	pin 26	Pin Mode (Predefined)	mode #	Generic PIN I/O Legal Settings																PIN_INP	0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	PIN_INP_HI	1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	PIN_INP_LO	2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	PIN_OUT	3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	PIN_OUT_OD	4	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	PIN_AN	5	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	PIN_ANAVG	6	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
4D Pin Name (Predefined)		PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7	PA8	PA9	PA10	PA11	PA12	PA13	PA14	PA15																																																																																																																																																																																							
DIABLO16 Pin Number		pin 61	pin 62	pin 63	pin 64	pin 46	pin 49	pin 50	pin 51	pin 52	pin 53	pin 43	pin 44	pin 31	pin 32	pin 37	pin 36																																																																																																																																																																																							
H1 Pin Number		pin 1	pin 3	pin 5	pin 7	pin 29	pin 27	pin 25	pin 23	pin 21	pin 19	pin 8	pin 6	pin 28	pin 30	pin 24	pin 26																																																																																																																																																																																							
Pin Mode (Predefined)	mode #	Generic PIN I/O Legal Settings																																																																																																																																																																																																						
PIN_INP	0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓																																																																																																																																																																																							
PIN_INP_HI	1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓																																																																																																																																																																																							
PIN_INP_LO	2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓																																																																																																																																																																																							
PIN_OUT	3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓																																																																																																																																																																																							
PIN_OUT_OD	4	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓																																																																																																																																																																																							
PIN_AN	5	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗																																																																																																																																																																																							
PIN_ANAVG	6	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗																																																																																																																																																																																							
Example	<pre>pin_Set(PIN_INP, PA0); // set PA0 to be an input pin_Set(PIN_AN, PA1); // set PA1 to be an Analog input pin_Set(PIN_INP_HI, PA4); // set PA4 to be an input with int. pullup pin_Set(PIN_INP_LO, PA5); // set PA5 to be an input with int. pulldown pin_Set(PIN_OUT, PA10); // set PA10 to be used as an output pin_Set(PIN_OUT_OD, PA14); // set PA14 to be an Open Drain Output pin_Set(PIN_ANAVG, PA0); // set PA0 to be an Averaging Analog Input</pre>																																																																																																																																																																																																							

2.1.2. pin_HI(pin)

Syntax	<code>pin_HI(pin);</code>																																																				
Arguments	pin																																																				
	pin	A value (usually a constant) specifying the pin number or a predefined pin name.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	Returns a Logic 1 (0x0001) if the pin number is legal.																																																			
Description	Set any pin to the HI state, pin is automatically made an output. Pullup, Pulldown, and change notification will be disabled for the selected pin.																																																				
	<table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>Yes</td></tr> <tr><td>PA15</td><td>36</td><td>Yes</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	Yes	PA15	36	Yes
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	Yes																																																			
PA15	36	Yes																																																			
Example	<code>pin_HI(PA7); // output a Logic 1 on PA7 pin</code>																																																				

2.1.3. pin_LO(pin)

Syntax	<code>pin_LO(pin);</code>																																																				
Arguments	pin																																																				
	pin	A value (usually a constant) specifying the pin number or a predefined pin name.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	Returns a Logic 1 (0x0001) if the pin number is legal.																																																			
Description	Set any pin to the LOW state, pin is automatically made an output. Pullup, Pulldown, and change notification will be disabled for the selected pin.																																																				
	<table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>Yes</td></tr> <tr><td>PA15</td><td>36</td><td>Yes</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	Yes	PA15	36	Yes
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	Yes																																																			
PA15	36	Yes																																																			
Example	<code>pin_LO(PA7); // output a Logic 0 on PA7 pin</code>																																																				

2.1.4. pin_Val(pin)

Syntax	<code>pin_Val(pin, value);</code>																																																				
Arguments	pin, value																																																				
	pin	A value (usually a constant) specifying the pin number or a predefined pin name.																																																			
	value	Bit 0 of value																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	Returns a Logic 1 (0x0001) if the pin number is legal.																																																			
Description	<p>Outputs a logic state on a pin depending on the value of bit 0 of a variable. The pin is automatically made an output. Pullup, Pulldown, and change notification will be disabled for the selected pins.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>Yes</td></tr> <tr><td>PA15</td><td>36</td><td>Yes</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	Yes	PA15	36	Yes
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	Yes																																																			
PA15	36	Yes																																																			
Example	<pre>temp := 1; pin_Val(PA4, temp); // output a Logic 4 on PA7 pin</pre>																																																				

2.1.5. pin_Read(pin)

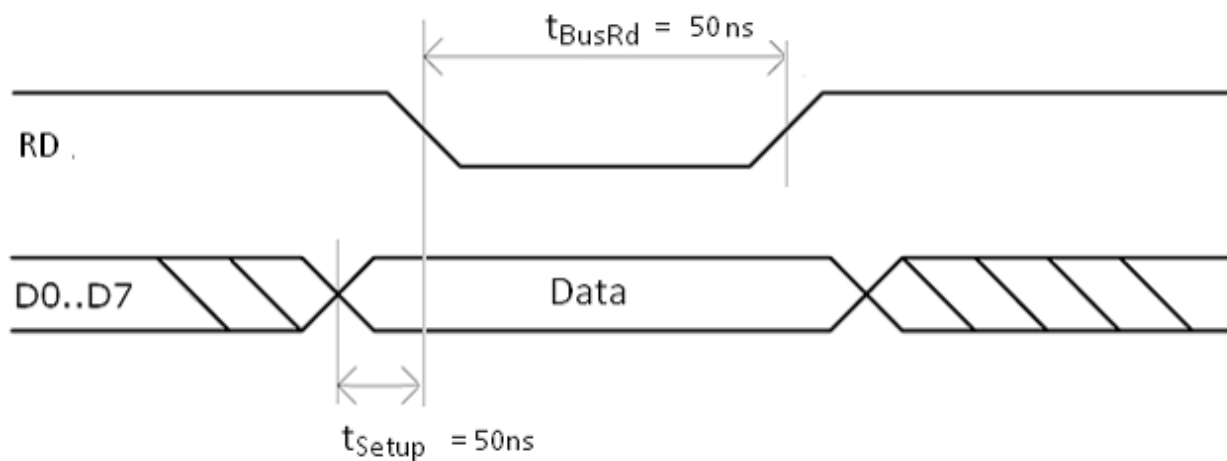
Syntax	<code>pin_Read(pin);</code>																																																				
Arguments	pin																																																				
	pin	A value (usually a constant) specifying the pin number or a predefined pin name.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	Returns state of the pin a Logic 0 (0x0001) or 1 (0x0001) if the pin is set to digital input. Returns state of the output latch, a Logic 0 (0x0001) or 1 (0x0001) if the pin is set to digital output. Returns 12 bit analogue value if the pin is set to an analogue pin.																																																			
Description	<p>Read a pin in various ways. If the pin is set to an input, read the state of the input pin. If set to an output, read the state of the output latch. If set to analogue, read the 12 bit analogue value.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>Yes</td></tr> <tr><td>PA15</td><td>36</td><td>Yes</td></tr> </tbody> </table> <p>When using PIN_AN or PIN_ANAVG via the pin_Set command, then please note: If Touch is enabled this function should be called no more than once per millisecond, otherwise touch behaviour could be erratic.</p> <p>PIN_AN > 15,000 reads/second</p> <p>PIN_ANAVG ~3,000 reads/second</p>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	Yes	PA15	36	Yes
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	Yes																																																			
PA15	36	Yes																																																			
Example	<pre>pin_Set(PA1, PIN_AN); // set PA1 to be used as an Analog input ANval := pin_Read(PA1); // Read the 12bit analog input</pre>																																																				

2.1.6. bus_Read()

Syntax	<code>bus_Read();</code>																																																				
Arguments	none																																																				
Returns	value																																																				
	value	Returns the 16 bit value of the bus.																																																			
Description	<p>Read the 16 bit port regardless of pin configurations. If a pin is configured as input or analogue, the pin is read directly as if it were a digital input. If a pin is configured as an output, the pin is also read directly, giving the output latch state.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>Yes</td></tr> <tr><td>PA15</td><td>36</td><td>Yes</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	Yes	PA15	36	Yes
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	Yes																																																			
PA15	36	Yes																																																			
Example	<pre>var1 := bus_Read(); //Read the 16bit value off PA0-PA15 pins</pre>																																																				

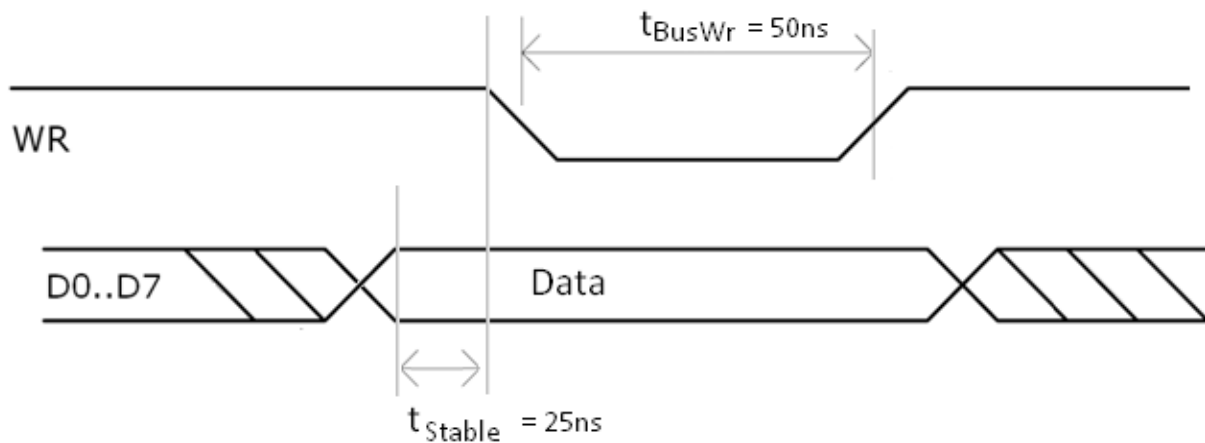
2.1.7. bus_Read8()

Syntax	<code>bus_Read8();</code>	
Arguments	none	
Returns	value	
	value	Returns the state of the 8 bit bus as an 8bit value.
Description	<p>Returns the state of the bus as an 8bit value in to the lower byte of the assigned variable.</p> <p>The BUS_RD pin set to LO, then, after a settling delay of approx 50nsec, the BUS is read into the lower 8 bits of the assigned variable (the upper 8 bits being set to 0) the BUS_RD pin is then set back to a HI level.</p> <p>Note: The BUS_RD pin must be preset to the desired output state must the bus pins to ensure BUS write integrity.</p> <p>BUS_RD is PA3</p> <p>The 8 bit BUS pins 0 to 7 are PA4 to PA11</p>	
Example	<pre>var1 := bus_Read8();</pre> <p>The lower byte of var1 will get loaded with the state of the bus.</p>	



2.1.8. bus_Write8(value)

Syntax	<code>bus_Write8(value);</code>	
Arguments	value	
	value	The lower 8 bits of value are sent to the 8 bit bus.
	The argument can be a variable, array element, expression or constant.	
Returns	nothing	
Description	<p>The lower 8 bits of arg1 are placed on the BUS, then, after a settling delay of approx 50nsec, the BUS_WR pin is strobed LO for approx 50nsec then set back HI.</p> <p>The upper 8 bits of arg1 are ignored.</p> <p>Note: The BUS_WR pin must be preset to the desired output state as must the bus pins to ensure BUS write integrity.</p> <p>BUS_WR is PA2</p> <p>The 8 bit BUS pins 0 to 7 are PA4 to PA11</p>	
Example	<pre>var data1 ; data1 := 0x05; bus_Write8(data1);</pre>	



2.1.9. bus_SetPins(value)

Syntax	<code>bus_SetPins(value);</code>																																																				
Arguments	value																																																				
	value	A value (usually a constant) specifying the pin number. Bit 0 corresponds to PA0 through to bit9 which corresponds to PA9.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	Nothing																																																				
Description	<p>Any '1' bits in "value" sets the corresponding port pin to an output and forces its state to a '1'. The state of its previous open drain configuration is not altered. Any '0' bits in "value" will not affect the pin. pullup, pulldown, and change notification will be disable for the selected pins.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>var arg1; arg1 := 0b0011010; // set desired mask bus_SetPins(arg1); // set PA1, PA3 and PA4 to output, making them HI</pre>																																																				

2.1.10. bus_ClearPins(value)

Syntax	<code>bus_ClearPins(value);</code>																																																				
Arguments	value																																																				
	value	A value (usually a constant) specifying the pin number. Bit 0 corresponds to PA0 through to bit9 which corresponds to PA9.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	Nothing																																																				
Description	Any '1' bits in "value" sets the corresponding port pin to an output and forces its state to a '0'. The state of its previous open drain configuration is not altered. Any '0' bits in "value" will not affect the pin. pullup, pulldown, and change notification will be disable for the selected pins.																																																				
	<table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>var arg1; arg1 := M_PA1 M_PA3 M_PA4 ; // set desired mask (same as 0b0011010) bus_ClearPins(arg1); // set PA1, PA3 and PA4 to output, making them LO</pre>																																																				

2.1.11. bus_SetChangeInterrupt (function, portmask)

Syntax	<code>bus_SetChangeInterrupt(function, portmask);</code>																																																				
Arguments	function, portmask																																																				
	function	Event Function to be queued when an interrupt occurs.																																																			
	portmask	"portmask" marks that pin to generate an interrupt on change. A value (usually a constant) specifying the pin number or a predefined pin name.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	Return the current state of the pins that are selected in "portmask". This can be saved and later used in "function" to see which pin(s) actually changed																																																			
Description	<p>Any '1' bits in "portmask" marks that pin to generate an interrupt on change. A level change on that pin will cause "function" to be executed. If "function" is zero, the display may be put into sleep mode, and any change will cause a wakeup reset. Wakeup will always re-start code running in FLASHBANK_0 Bit 0 corresponds to PA0 through to bit15 which corresponds to PA15</p> <p>Once armed, "function" will only be executed once, it is necessary to re-arm for any further events.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>Yes</td></tr> <tr><td>PA15</td><td>36</td><td>Yes</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	Yes	PA15	36	Yes
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	Yes																																																			
PA15	36	Yes																																																			
Example	<pre>bus_SetChangeInterrupt(scanKeypad, M_PA4 M_PA5 M_PA6 M_PA7); // set PA4 to PA7 to interrupt on change</pre>																																																				

2.1.12. Qencoder1(PHApin, PHBpin, mode)

Syntax	Qencoder1(PHApin, PHBpin, mode);																																																				
Arguments	PHApin, PHBpin, mode																																																				
	PHApin	Phase A input pin, 4D Pin Name reference – see table below																																																			
	PHBpin	Phase B input pin, 4D Pin Name reference – see table below																																																			
	mode	Not currently used, set to 0 only.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	Nothing																																																				
Description	<p>Connect a quadrature encoder to a pair of pins, using the predefined 4D Pin Names in the table below, and the PHApin and PHBpin arguments in this function.</p> <p>It is necessary to configure the pins first, depending on your requirements, e.g. <code>pin_Set(PIN_INP_HI, PA4); // PA4 as input, with pullup to Vcc</code> or maybe <code>pin_Set(PIN_INP, PA4); // PA4 as input, no pullup or pulldown</code></p> <p>The position counter and delta can be read or written to at any time with peekW and pokeW using the following constants:</p> <p>QEN1_COUNTER_LO QEN1_COUNTER_HI QEN1_DELTA</p> <p>QEN1_DELTA is reset to 0 once it has been read</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>var qen1Delta; pin_Set(PIN_INP_HI, PA4); // Set PA4 to be Input with Pullup pin_Set(PIN_INP_HI, PA5); // Set PA5 to be Input with Pullup Qencoder1(PA4, PA5, 0); // connect PA4 and PA5 pins to quadrature encoder module #1 qen1Delta := peekW(QEN1_DELTA);</pre>																																																				

2.1.13. Qencoder1Reset()

Syntax	Qencoder1Reset();
Arguments	None
Returns	Nothing
Description	Resets the Counters and Delta values for Encoder #1 QEN1_COUNTER_LO is reset to zero QEN1_COUNTER_HI is reset to zero QEN1_DELTA is reset to zero
Example	<pre>Qencoder1Reset(); // Reset the Counter and Delta values</pre>

2.1.14. Qencoder2(PHApin, PHBpin, mode)

Syntax	Qencoder2(PHApin, PHBpin, mode);																																																				
Arguments	PHApin, PHBpin, mode																																																				
	PHApin	Phase A input pin, 4D Pin Name reference – see table below																																																			
	PHBpin	Phase B input pin, 4D Pin Name reference – see table below																																																			
	mode	Not currently used, set to 0 only.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	Nothing																																																				
Description	<p>Connect a quadrature encoder to a pair of pins, using the predefined 4D Pin Names in the table below, and the PHApin and PHBpin arguments in this function.</p> <p>It is necessary to configure the pins first, depending on your requirements, e.g. <code>pin_Set(PIN_INP_HI, PA8); // PA8 as input, with pullup to Vcc</code> or maybe <code>pin_Set(PIN_INP, PA9); // PA9 as input, no pullup or pulldown</code></p> <p>The position counter and delta can be read or written to at any time with peekW and pokeW using the following constants:</p> <p>QEN2_COUNTER_LO QEN2_COUNTER_HI QEN2_DELTA</p> <p>QEN2_DELTA is reset to 0 once it has been read</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>var qen2Delta; pin_Set(PIN_INP, PA8); // Set PA8 to be Input pin_Set(PIN_INP, PA9); // Set PA9 to be Input Qencoder2(PA8, PA9, 0); // connect PA8 and PA9 pins to quadrature encoder module #2 pokeW(QEN2_COUNTER_HI) := 12; // some 'preset value'</pre>																																																				

2.1.15. Qencoder2Reset()

Syntax	<code>Qencoder2Reset();</code>
Arguments	None
Returns	Nothing
Description	Resets the Counters and Delta values for Encoder #2 QEN2_COUNTER_LO is reset to zero QEN2_COUNTER_HI is reset to zero QEN2_DELTA is reset to zero
Example	<code>Qencoder2Reset(); // Reset the Counter and Delta values</code>

2.1.16. pwm_Init(pin, mode, value)

Syntax	<code>pwm_Init(pin, mode, value);</code>																																																														
Arguments	<code>pin, mode, value</code>																																																														
	Pin	4D Pin Name to enable the PWM on																																																													
	mode	Modes for the PWM, see description below																																																													
	value	Value determines Duty Cycle/Time Base depending on Mode, see below																																																													
	The arguments can be a variable, array element, expression or constant.																																																														
Returns	Status	Returns TRUE if the pin number is legal, usually ignored																																																													
Description	<p>This PWM function enables a PWM output on the desired pin, based on the availability set out by the table below.</p> <p>Set the pin using the predefined 4D Pin Name into the pin argument, and select its mode and value, which are determined by:</p> <table border="1"> <thead> <tr> <th>PWM Mode</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>PWM_OFF (0)</td> <td>Turn off the PWM (pin is left as Output)</td> </tr> <tr> <td>PWM_PLAIN (1)</td> <td>Plan PWM which value is a number between 0 and 1000. This corresponds to a 0.0 to 100.0% duty cycle. Raw Frequency is ~70kHz. A value of 1 is not valid.</td> </tr> <tr> <td>PWM_SERVO (2)</td> <td>Servo PWM has a value which is between 100 and 200. This corresponds to 1.00 to 2.00ms. Please note values from 0 to 600 are valid (0-6ms), but should be used with caution. Repetition Rate is ~50Hz or 20ms</td> </tr> <tr> <td>PWM_BINARY (3)</td> <td>Binary PWM which value is a number between 0 and 1024. This corresponds to a 0.0 to 100.0% duty cycle. Raw Frequency is ~68kHz. A value of 1 is not valid.</td> </tr> </tbody> </table> <p>The <code>pwm_Init</code> is non-blocking and the <code>pwm</code> continues until turned off</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>No</td></tr> <tr><td>PA1</td><td>62</td><td>No</td></tr> <tr><td>PA2</td><td>63</td><td>No</td></tr> <tr><td>PA3</td><td>64</td><td>No</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>No</td></tr> <tr><td>PA11</td><td>44</td><td>No</td></tr> <tr><td>PA12</td><td>31</td><td>No</td></tr> <tr><td>PA13</td><td>32</td><td>No</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		PWM Mode	Description	PWM_OFF (0)	Turn off the PWM (pin is left as Output)	PWM_PLAIN (1)	Plan PWM which value is a number between 0 and 1000. This corresponds to a 0.0 to 100.0% duty cycle. Raw Frequency is ~70kHz. A value of 1 is not valid.	PWM_SERVO (2)	Servo PWM has a value which is between 100 and 200. This corresponds to 1.00 to 2.00ms. Please note values from 0 to 600 are valid (0-6ms), but should be used with caution. Repetition Rate is ~50Hz or 20ms	PWM_BINARY (3)	Binary PWM which value is a number between 0 and 1024. This corresponds to a 0.0 to 100.0% duty cycle. Raw Frequency is ~68kHz. A value of 1 is not valid.	4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	No	PA1	62	No	PA2	63	No	PA3	64	No	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	No	PA11	44	No	PA12	31	No	PA13	32	No	PA14	37	No	PA15	36	No
PWM Mode	Description																																																														
PWM_OFF (0)	Turn off the PWM (pin is left as Output)																																																														
PWM_PLAIN (1)	Plan PWM which value is a number between 0 and 1000. This corresponds to a 0.0 to 100.0% duty cycle. Raw Frequency is ~70kHz. A value of 1 is not valid.																																																														
PWM_SERVO (2)	Servo PWM has a value which is between 100 and 200. This corresponds to 1.00 to 2.00ms. Please note values from 0 to 600 are valid (0-6ms), but should be used with caution. Repetition Rate is ~50Hz or 20ms																																																														
PWM_BINARY (3)	Binary PWM which value is a number between 0 and 1024. This corresponds to a 0.0 to 100.0% duty cycle. Raw Frequency is ~68kHz. A value of 1 is not valid.																																																														
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																													
PA0	61	No																																																													
PA1	62	No																																																													
PA2	63	No																																																													
PA3	64	No																																																													
PA4	46	Yes																																																													
PA5	49	Yes																																																													
PA6	50	Yes																																																													
PA7	51	Yes																																																													
PA8	52	Yes																																																													
PA9	53	Yes																																																													
PA10	43	No																																																													
PA11	44	No																																																													
PA12	31	No																																																													
PA13	32	No																																																													
PA14	37	No																																																													
PA15	36	No																																																													
Example	<code>pwm_Init(PA4, PWM_PLAIN, 676); //Sets Plain PWM of 67.7% on PA4</code>																																																														

2.1.17. pin_Pulseout(pin, value)

Syntax	<code>pin_Pulseout(pin, value);</code> or <code>pin_PulseoutB(pin, value)</code>																																																				
Arguments	pin, value																																																				
	Pin	4D predefined Pin Name to enable Pulseout on																																																			
	value	Length of pulse in milliseconds																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	Returns TRUE if the pin number is legal (usually ignored)																																																				
Description	<p>This function will invert the state of an output for "value" milliseconds.</p> <p>pin_Pulseout is a non-Blocking function, that is, code execution may continue while a pulse is occurring, and pulses can occur on multiple pins simultaneously.</p> <p>pin_PulseoutB is a Blocking function, where program execution is suspended during pulse.</p> <p>If not already an output, pin is automatically made a push/pull output, and the last state of its output latch will determine pulse polarity.</p> <p>Its open drain state is not altered if the pin was already an output.</p> <p>If pulseout is called while pulseout is still active, the pulse timer will simply be updated with the new "value" and the pulse will continue with the extended value.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>No</td></tr> <tr><td>PA11</td><td>44</td><td>No</td></tr> <tr><td>PA12</td><td>31</td><td>No</td></tr> <tr><td>PA13</td><td>32</td><td>No</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	No	PA11	44	No	PA12	31	No	PA13	32	No	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	No																																																			
PA11	44	No																																																			
PA12	31	No																																																			
PA13	32	No																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>pin_Pulseout(PA3, 105); // create a Hi Pulse of 105ms on PA3 ... pin_set(PIN_OUT, PA1); // set PA1 as an Output pin_HI(PA1); // set PA1 to output HI pin_Pulseout(PA1, 50); // create a Lo pulse of 50ms on PA1</pre>																																																				

2.1.18. pin_Counter(pin, mode, OVfunction)

Syntax	pin_Counter(pin, mode, OVfunction);																																																														
Arguments	pin, mode, OVfunction																																																														
	pin	4D predefined Pin Name to enable pin counter on, see table below																																																													
	mode	Counter mode, see table below																																																													
	OVfunction	Event function to be queued on overflow of counter																																																													
	The arguments can be a variable, array element, expression or constant.																																																														
Returns	Nothing																																																														
Description	<p>Connect a counter to a pin to count transistions, and optionally call an event function when the 16bit counter wraps from 0xFFFF to zero.</p> <p>The counter can be read or written to at any time with peekW and pokeW, therefore, the count may be set to 0xFFFF for example, so that user function "OVfunction" will be called after 16 pulses.</p> <p>If "OVfunction" is set to zero, only the counter will increment, and simply wrap back to zero from 0xFFFF. If "OVfunction" points to a user function, when the event fires, pin_Counter will be disabled, and will need to be re-armed (ie '1shot' operation)</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>No</td></tr> <tr><td>PA1</td><td>62</td><td>No</td></tr> <tr><td>PA2</td><td>63</td><td>No</td></tr> <tr><td>PA3</td><td>64</td><td>No</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>No</td></tr> <tr><td>PA11</td><td>44</td><td>No</td></tr> <tr><td>PA12</td><td>31</td><td>No</td></tr> <tr><td>PA13</td><td>32</td><td>No</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table> <p>The pin may be configured as an input or output, the function behaves the same.</p> <p>All six pin counters may be active simultaneously, and the maximum frequency of pin transistions should not exceed a few Khz in mode 1 and 2 and are usually used for simple process control counting.</p> <table border="1"> <thead> <tr> <th>Pin Counter MODE</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>COUNT_OFF (0)</td> <td>Disconnect the counter from the pin, "OVfunction" is therefore ignored, and counting is inhibited.</td> </tr> <tr> <td>COUNT_RISE (1)</td> <td>increment counter on every rising edge</td> </tr> <tr> <td>COUNT_FALL (2)</td> <td>increment on every falling edge</td> </tr> <tr> <td>COUNT_EDGE (3)</td> <td>increment on every rising and falling edge</td> </tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	No	PA1	62	No	PA2	63	No	PA3	64	No	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	No	PA11	44	No	PA12	31	No	PA13	32	No	PA14	37	No	PA15	36	No	Pin Counter MODE	Description	COUNT_OFF (0)	Disconnect the counter from the pin, "OVfunction" is therefore ignored, and counting is inhibited.	COUNT_RISE (1)	increment counter on every rising edge	COUNT_FALL (2)	increment on every falling edge	COUNT_EDGE (3)	increment on every rising and falling edge
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																													
PA0	61	No																																																													
PA1	62	No																																																													
PA2	63	No																																																													
PA3	64	No																																																													
PA4	46	Yes																																																													
PA5	49	Yes																																																													
PA6	50	Yes																																																													
PA7	51	Yes																																																													
PA8	52	Yes																																																													
PA9	53	Yes																																																													
PA10	43	No																																																													
PA11	44	No																																																													
PA12	31	No																																																													
PA13	32	No																																																													
PA14	37	No																																																													
PA15	36	No																																																													
Pin Counter MODE	Description																																																														
COUNT_OFF (0)	Disconnect the counter from the pin, "OVfunction" is therefore ignored, and counting is inhibited.																																																														
COUNT_RISE (1)	increment counter on every rising edge																																																														
COUNT_FALL (2)	increment on every falling edge																																																														
COUNT_EDGE (3)	increment on every rising and falling edge																																																														

Example	<pre>func main() pin_Set(PIN_INP, PA4); // external start event repeat // main loop if(pin_Read(PA4)) pin_Counter(PA2, COUNT_RISE, userFunc); endif // user code here forever endfunc func userFunc() print("Hello World"); endfunc</pre>
----------------	--

2.1.19. ana_HS(rate, samples, IO1buf, IO2buf, IO3buf, IO4buf, userFunction)

Syntax	<code>ana_HS(rate, samples, IO1buf, IO2buf, IO3buf, IO4buf, userFunction);</code>	
Arguments	<code>rate, samples, IO1buf, IO2buf, IO3buf, IO4buf, userFunction</code>	
	rate	Number of samples per second, see rate commend below
	samples	Number of samples to collect per analog channel
	IO1buf	Buffer Address for first Analog Channel
	IO2buf	Buffer Address for second Analog Channel
	IO3buf	Buffer Address for third Analog Channel
	IO4buf	Buffer Address for forth Analog Channel
	userFunction	Function to call once all samples have been collected
	The arguments can be a variable, array element, expression or constant.	
Returns	Nothing	
Description	<p>Collects "samples" samples at "rate" frequency for 0 to 4 analogue pins and calls "userFunction" when done.</p> <p>"rate" is samples represented as 1/100 samples per second, up to 250,000 reads/second across 1-4 channels. For example if you wish to sample at 5000 samples per second, you would set rate to be 50 as $5000 * 1/100 = 50$.</p> <p>Any unused IOx pins should have their buffer addresses (i.e. IO4buf) set to 0</p> <p>For performance reasons samples are taken in chunks of 32, thus if you request 33 samples there will be a delay of 31 samples before "userFunction" is called</p> <p>Note: If Touch is enabled this function should be called no more than once per millisecond, otherwise touch behaviour could be erratic.</p>	
Example	<pre>var x[100]; // Buffer for IO1buf var b[100]; // Buffer for IO2buf var c[100]; // Buffer for IO3buf // 1000 samples a second, 10000 samples to be collected from 3 channels ana_HS(1000, 10, a, b, c, 0, myFunc); func myFunc() //do something once samples collected Endfunc</pre>	

2.1.20. pin_PulseoutCount(pin, frequency, count, function)

Syntax	<code>pin_PulseoutCount(pin, frequency, count, function);</code>																																																				
Arguments	pin, frequency, count, function																																																				
	pin	4D predefined Pin Name to enable PulseoutCount on																																																			
	frequency	The frequency to pulse the pin at (minimum 10Hz)																																																			
	count	The number of times to pulse the specified pin																																																			
	function	Address of a function to be called at completion																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	Returns TRUE if the pin number is legal and the frequency is at least 10Hz and the maximum number of 3 simultaneous pulseoutCount pins is not exceeded																																																				
Description	<p>This function will invert the state of an output at a "freq" frequency "count" times. This is a non-Blocking function, that is, code execution may continue while a pulse is occurring, and pulses can occur on multiple pins simultaneously. A function can be specified that will be called when all the pulses have been output. A maximum of 3 pulseoutCount activities can be active at any one point.</p> <p>If not already an output, pin is automatically made a push/pull output, and the last state of its output latch will determine pulse polarity.</p> <p>Its open drain state is not altered if the pin was already an output.</p> <p>If pulseoutCount is called while pulseoutCount is active, the pulse counter will simply have the new count value added to it.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>No</td></tr> <tr><td>PA1</td><td>62</td><td>No</td></tr> <tr><td>PA2</td><td>63</td><td>No</td></tr> <tr><td>PA3</td><td>64</td><td>No</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>No</td></tr> <tr><td>PA11</td><td>44</td><td>No</td></tr> <tr><td>PA12</td><td>31</td><td>No</td></tr> <tr><td>PA13</td><td>32</td><td>No</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	No	PA1	62	No	PA2	63	No	PA3	64	No	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	No	PA11	44	No	PA12	31	No	PA13	32	No	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	No																																																			
PA1	62	No																																																			
PA2	63	No																																																			
PA3	64	No																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	No																																																			
PA11	44	No																																																			
PA12	31	No																																																			
PA13	32	No																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>pin_Pulseout(PA3, 105); // create a Hi Pulse of 105ms on PA3 ... pin_set(PIN_OUT, PA1); // set PA1 as an Output pin_HI(PA1); // set PA1 to output HI pin_Pulseout(PA1, 50); // create a Lo pulse of 50ms on PA1</pre>																																																				

2.1.21. OW_Reset(pin)

Syntax	OW_Reset(pin);		
Arguments	pin		
	pin	4D predefined Pin Name, see table below.	
	The arguments can be a variable, array element, expression or constant.		
Returns	result		
	result	Reset, and returns the status of the ONEWIRE device 0 = ACK 1 = No Activity (refer to Dallas 1wired documentation for further information)	
Description	Resets a ONEWIRE device and returns the status.		
	4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability
	PA0	61	Yes
	PA1	62	Yes
	PA2	63	Yes
	PA3	64	Yes
	PA4	46	Yes
	PA5	49	Yes
	PA6	50	Yes
	PA7	51	Yes
	PA8	52	Yes
	PA9	53	Yes
	PA10	43	Yes
	PA11	44	Yes
	PA12	31	Yes
	PA13	32	Yes
	PA14	37	No
	PA15	36	No
Example	<pre>print ("result=", OW_Reset(PA0));</pre> <p>This example will print a 0 if the device initialised successfully.</p>		

2.1.22. OW_Read(pin)

Syntax	OW_Read(pin);																																																				
Arguments	pin																																																				
	Pin	4D predefined Pin Name, see table below.																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	A word holding the lower 8 bits contain data bits received from the 1-Wire device.																																																			
Description	<p>Reads the 8 bit value from a 1-Wire devices register. (refer to Dallas 1wired documentation for further information)</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>// read temperature from DS1821 device var temp_buf; OW_Reset(PA0); // reset the device OW_Write(PA0,0xAA); // send the read command temp_buf := OW_Read(PA0); // read the device register</pre>																																																				

2.1.23. OW_Read9(pin)

Syntax	OW_Read9(pin);																																																					
Arguments	pin																																																					
	Pin	4D predefined Pin Name, see table below.																																																				
	The arguments can be a variable, array element, expression or constant.																																																					
Returns	value																																																					
	value	A word holding 9 or more data bits received from the 1-Wire device.																																																				
Description	<p>Reads the 9 or more bit value from a 1-Wire devices register. (refer to Dallas 1wired documentation for further information)</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>			4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																				
PA0	61	Yes																																																				
PA1	62	Yes																																																				
PA2	63	Yes																																																				
PA3	64	Yes																																																				
PA4	46	Yes																																																				
PA5	49	Yes																																																				
PA6	50	Yes																																																				
PA7	51	Yes																																																				
PA8	52	Yes																																																				
PA9	53	Yes																																																				
PA10	43	Yes																																																				
PA11	44	Yes																																																				
PA12	31	Yes																																																				
PA13	32	Yes																																																				
PA14	37	No																																																				
PA15	36	No																																																				
Example	<pre>// read temperature from DS1821 device var temp_buf; OW_Reset(PA0); // reset the device OW_Write(PA0,0xAA); // send the read command temp_buf := OW_Read9(PA0); // read the device register</pre>																																																					

2.1.24. OW_Write(pin, data)

Syntax	OW_Write(pin, data);		
Arguments	pin, data		
	Pin	4D predefined Pin Name, see table below.	
	Data	The lower 8 bits of data are sent to the 1-Wire device.	
	The argument can be a variable, array element, expression or constant.		
Returns	Nothing		
Description	Writes the 8 bit data to 1-Wire devices register. (refer to Dallas 1wired documentation for further information)		
	4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability
	PA0	61	Yes
	PA1	62	Yes
	PA2	63	Yes
	PA3	64	Yes
	PA4	46	Yes
	PA5	49	Yes
	PA6	50	Yes
	PA7	51	Yes
	PA8	52	Yes
	PA9	53	Yes
	PA10	43	Yes
	PA11	44	Yes
	PA12	31	Yes
	PA13	32	Yes
	PA14	37	No
	PA15	36	No
Example	<pre>//===== // For this demo to work, a Dallas DS18B20 must be connected to // PA0 AND POWERED FROM 3.3 to 5V. // DS18B20 pin1 = Gnd / pin2 = data in/out / pin 3 = +3.3v // Refer to the Dallas DS18B20 for further information //===== func main() var temp_buf ; pause(1000); txt_MoveCursor(0,0); if(OW_Reset(PA0)) // initialise and test print("No device detected"); while(1); endif repeat txt_MoveCursor(0, 0); print ("result=", OW_Reset(PA0)); OW_Write(PA0, 0xcc); // skip ROM OW_Write(PA0, 0x44); // start conversion OW_Reset(PA0); // reset OW_Write(PA0, 0xcc); // skip ROM OW_Write(PA0, 0xBE); // get temperature temp_buf := OW_Read(PA0); temp_buf += (OW_Read(PA0) << 8);</pre>		

```
        txt_MoveCursor(1, 0);
        print ("temp_buf=0x", [HEX4] temp_buf);
    forever
endfunc
```

2.1.25. NP_Write(pin, data, size, Options, RepeatFirst, Repeat, RepeatLast)

Syntax	NP_Write(pin, data, size, Options, RepeatFirst, Repeat, RepeatLast) ;																																																				
Arguments	pin, data, size, Options, RepeatFirst, Repeat, RepeatLast																																																				
	Pin	4D predefined Pin Name, see table below.																																																			
	data	The address of the data to be sent																																																			
	size	The size of the data to be sent, in Pixels																																																			
	Options	The format of the data pixels, NP_565, NP_RGB or NP_XRGB																																																			
	RepeatFirst	Number of times to repeat the first colour (0 means first colour is not considered 'special')																																																			
	Repeat	Number of times to repeat the colours between first and last																																																			
	RepeatLast	Number of times to repeat the last colour (0 means last colour is not considered 'special')																																																			
	The arguments can be a variable, array element, expression or constant.																																																				
Returns	value																																																				
	value	Returns TRUE if the pin number is legal (usually ignored)																																																			
Description	<p>Writes a string of pixels to the NeoPixel array connected to the specified I/O Pin.</p> <p>Due to the critical timing requirements of the NeoPixel, any interrupts should be stopped, or otherwise 'circumvented' before this command is issued. Internally, the system Timer is disabled during this command.</p> <p>Comms Interrupts should also be disabled by the user, otherwise errors may occur. A suitable workaround is to repeat the NP_Write until 'com_Count' does not change during its execution.</p> <p>Comms TX Buffers, if used, should be held.</p> <p>Audio should be stopped or paused.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<pre>var data[4] := [RED, LIME, BLUE, WHITE] ; // send Red, Lime Blue, and white to the NeoPixel strip twice</pre>																																																				


```
NP_Write(PA0, data, 4, 0, 2, 0);  
// send 2 x Red, Lime, Blue and 2 x White to the NeoPixel strip  
NP_Write(PA0, data, 4, 2, 1, 2);
```

2.2. System Memory Access Functions

Summary of Functions in this section:

- peekW(address)
- pokeW(address, word_value)

2.2.1. peekW(address)

Syntax	<code>peekW(address);</code>	
Arguments	address	
	address	The address of a memory word. The address is usually a pre-defined system register address constant, (see the address constants for all the system word sized registers in section 3).
	The arguments can be a variable, array element, expression or constant.	
Returns	word_value	
	word_value	The 16 bit value stored at address .
Description	Read a word from system memory. Note: that the txt_Set variables (0-15) and gfx_set variables (16-31) can also be accessed with peekW and pokeW.	
Example	<pre>var myvar; myvar := peekW(SYSTEM_TIMER_LO);</pre> <p>This example places the low word of the 32 bit system timer in myvar.</p>	

2.2.2. pokeW(address, word_value)

Syntax	<code>pokeW(address, word_value);</code>	
Arguments	<code>address, word_value</code>	
	address	The address of a memory word. The address is usually a pre-defined system register address constant, (see the address constants for all the system word sized registers in section 3).
	word_value	The 16 bit word_value will be stored at address .
	The arguments can be a variable, array element, expression or constant.	
Returns	None	
Description	This function writes a 16 bit value to a location specified by address . Note: that the txt_Set variables (0-15) and gfx_set variables (16-31) can also be accessed with peekW and pokeW.	
Example	<code>pokeW(TIMER2, 5000);</code> This example sets TIMER2 to 5 seconds.	

2.3. Maths Functions

Summary of Functions in this section:

- ABS(value)
- MIN(value1, value2)
- MAX(value1, value2)
- SWAP(&var1, &var2)
- SIN(angle)
- COS(angle)
- RAND()
- RANDVAL(low, high)
- SEED(number)
- SQRT(number)
- OVF ()
- CY()
- EVE_SP()
- EVE_SSIZE()
- umul_1616(&res32, val1, val2)
- uadd_3232(&res32, &val1, &val2)
- usub_3232(&res32, &val1, &val2)
- udiv_3232(&res32, &var1, &var2)
- ucmp_3232(&val1, &val2)

2.3.1. ABS(value)

Syntax	ABS(value);	
Arguments	value	
	value	A variable, array element, expression or constant.
	The arguments can be a variable, array element, expression or constant.	
Returns	value	
	value	Returns the absolute value.
Description	This function returns the absolute value of value .	
Example	<pre>var myvar, number; number := -100; myvar := ABS(number * 5);</pre> <p>This example returns 500 in variable myvar.</p>	

2.3.2. MIN(value1, value2)

Syntax	MIN(value1, value2);	
Arguments	value1, value2	
	value1	A variable, array element, expression or constant.
	value2	A variable, array element, expression or constant.
	The arguments can be a variable, array element, expression or constant.	
Returns	value	
	value	The smaller of the two values.
Description	This function returns the the smaller of value1 and value2 .	
Example	<pre>var myvar, number1, number2; number1 := 33; number2 := 66; myvar := MIN(number1, number2);</pre>	
	This example returns 33 in variable myvar .	

2.3.3. MAX(value1, value2)

Syntax	MAX(value1, value2);	
Arguments	value1, value2	
	value1	A variable, array element, expression or constant.
	value2	A variable, array element, expression or constant.
	The arguments can be a variable, array element, expression or constant.	
Returns	value	
	value	The larger of the two values.
Description	This function returns the larger of value1 and value2 .	
Example	<pre>var myvar, number1, number2; number1 := 33; number2 := 66; myvar := MAX(number1, number2);</pre>	
	This example returns 66 in variable myvar .	

2.3.4. SWAP(&var1, &var2)

Syntax	SWAP(&value1, &value2);	
Arguments	&var1, &var2	
	&var1	The address of the first variable.
	&var2	The address of the second variable.
	The arguments can only be a variable or an array element.	
Returns	nothing	
Description	Given the addresses of two variables (var1 and var2), the values at these addresses are swapped.	
Example	<pre>var number1, number2; number1 := 33; number2 := 66; SWAP(&number1, &number2);</pre> <p>This example swaps the values in number1 and number2. After the function is executed, number1 will hold 66, and number2 will hold 33.</p>	

2.3.5. SIN(angle)

Syntax	SIN(angle);	
Arguments	angle	
	angle	The angle in degrees. (Note: The input value is automatically shifted to lie within 0-359 degrees)
	The arguments can be a variable, array element, expression or constant.	
Returns	result	
	result	The sine in radians of an argument specified in degrees. The returned value range is from 127 to -127 which is a more useful representation for graphics work. The real sine values vary from 1.0 to -1.0 so appropriate scaling must be done in user code as required.
Description	This function returns the SIN of an angle	
Example	<pre>var myvar, angle; angle := 133; myvar := SIN(angle);</pre> <p>This example returns 92 in variable myvar.</p>	

2.3.6. COS(angle)

Syntax	COS(angle);	
Arguments	angle	
	angle	The angle in degrees. (Note: The input value is automatically shifted to lie within 0-359 degrees)
	The arguments can be a variable, array element, expression or constant.	
Returns	result	
	result	The cosine in radians of an argument specified in degrees. The returned value range is from 127 to -127 which is a more useful representation for graphics work. The real sine values vary from 1.0 to -1.0 so appropriate scaling must be done in user code as required.
Description	This function returns the COSINE of an angle	
Example	<pre>var myvar, angle; angle := 133; myvar := COS(angle);</pre> This example returns -86 in variable myvar .	

2.3.7. RAND()

Syntax	RAND();	
Arguments	none	
Returns	value	Returns a pseudo random signed number ranging from -32768 to +32767 each time the function is called. The random number generator may first be seeded by using the SEED(number) function. The seed will generate a pseudo random sequence that is repeatable. You can use the modulo operator (%) to return a number within a certain range, eg n := RAND() % 100; will return a random number between -99 and +99. If you are using random number generation for random graphics points, or only require a positive number set, you will need to use the ABS function so only a positive number is returned, eg: X1 := ABS(RAND() % 100); will set co-ordinate X1 between 0 and 99. Note that if the random number generator is not seeded, the first number returned after reset or power up will be zero. This is normal behavior.
Description	This function returns a pseudo random signed number ranging from -32768 to +32767	
Example	<pre>SEED(1234); print(RAND(), " ", RAND());</pre> <p>This example will print 3558, 1960 to the display.</p>	

2.3.8. RANDVAL(low, high)

Syntax	RANDVAL(low, high);	
Arguments	low, high	
	low	Low limit for the random numbers
	high	High limit for the random numbers
Returns	value	
	value	A random number between low and high limits.
	The arguments can be a variable, array element, expression or constant.	
Description	<p>Returns a random number between low and high limits such that $low \leq N < high$. The random number generator may first be seeded by using the SEED(number) function.</p> <p>RANDVAL is the equivalent of aggregate functions:- $myvar = ABS((RAND()%(high-low)+low));$</p> <p>Note: The lower limit is inclusive, but the upper limit is exclusive. Note: If the random number generator is not seeded, the first number returned after reset or power up will be the low number in the range. This is normal behaviour.</p>	
Example	<pre>SEED(1234); print(RAND(), " ", RAND());</pre> <p>This example will print 3558, 1960 to the display.</p>	

2.3.9. SEED(number)

Syntax	SEED(number);	
Arguments	number	
	number	Specifies the seed value for the pseudo random number generator.
	The arguments can be a variable, array element, expression or constant.	
Returns	nothing	
Description	This function seeds the pseudo random number generator so it will generate a new repeatable sequence. The seed value can be a positive or negative number.	
Example	<pre>SEED(-50); print(RAND()," ",RAND());</pre>	
	This example will print 30129, 27266 to the display.	

2.3.10. SQRT(number)

Syntax	SQRT(number);	
Arguments	number	
	number	Specifies the positive number for the SQRT function.
	The arguments can be a variable, array element, expression or constant.	
Returns	value	
	value	This function returns the integer square root which is the greatest integer less than or equal to the square root of number .
Description	This function returns the integer square root of a number.	
Example	<pre>var myvar; myvar := SQRT(26000);</pre>	
	This example returns 161 in variable myvar which is the integer square root of 26000.	

2.3.11. OVF()

Syntax	OVF();	
Arguments	none	
Returns	value	
	value	The high order 16 bits from certain math and shift functions.
Description	<p>This function returns the high order 16 bits from certain math and shift functions. It is extremely useful for calculating 32 bit address offsets for MEDIA access.</p> <p>It can be used with the shift operations, addition, subtraction, multiplication and modulus operations.</p>	
Example	<pre>var loWord, hiWord; loWord := 0x2710 * 0x2710; // (10000 * 10000 in hex format) hiWord := OVF(); print ("0x", [HEX] hiWord, [HEX] loWord);</pre> <p>This example will print 0x05F5E100 to the display , which is 100,000,000 in hexadecimal</p>	

2.3.12. CY()

Syntax	CY();	
Arguments	none	
Returns	Status	
	Status	Returns Status of carry, 0 or 1.
Description	This function returns the carry status of an unsigned overflow from any 16 or 32bit additions or subtractions.	
Example	<pre>var myvar; myvar := 0xFFFF8 + 9; // result = 1 print("myvar ", myvar, "\nCarry ", CY(), "\n"); // carry = 1</pre> <p>This example will print myvar 1 Carry 1</p>	

2.3.13. EVE_SP()

Syntax	EVE_SP();	
Arguments	None	
Returns	value	
	value	Returns the current stack level.
Description	Used for debugging to assess the current stack level, mainly for checking stack leaks.	
Example	<pre>var val; val := EVE_SP();</pre>	

2.3.14. EVE_SSIZE()

Syntax	EVE_SSIZE();	
Arguments	None	
Returns	value	
	value	Returns the stack size.
Description	Used to get the current stack size. Mainly for debugging purposes.	
Example	<pre>print(EVE_SSIZE());</pre> Prints stack size on the screen.	

2.3.15. `uadd_3232(&res32, &val1, &val2)`

Syntax	<code>uadd_3232(&res32, &val1, &val2);</code>	
Arguments	<code>&res32, &val1, &val2</code>	
	&res32	Points to 32bit result register.
	&val1	points to 32bit augend
	&val2	points to 32bit addend
Returns	value	
	value	Returns 1 on 32bit unsigned overflow (carry). Carry flag is also set on 32bit unsigned overflow and can be read with the <code>CY()</code> function.
Description	Performs an unsigned addition of 2 x 32bit values placing the 32bit result in a 2 word array.	
Example	<pre>var carry, valA[2], valB[2], Result[2]; var p; valA[0] := 0; valA[1] := 1; valB[0] := 0; valB[1] := 1; carry := uadd_3232(Result, valA, valB); p := str_Ptr(Result); print("0x"); str_Printf(&p, "%lX"); //prints the value at pointer in Hex long format.</pre> <p>This example will print 0x20000</p>	

2.3.16. usub_3232(&res32, &val1, &val2)

Syntax	<code>usub_3232(&res32, &val1, &val2);</code>	
Arguments	&res32, &val1, &val2	
	&res32	Points to 32bit result register.
	&val1	points to 32bit minuend
	&val2	points to 32bit subtrahend
Returns	Value	
	Value	Returns 1 on 32bit unsigned overflow (carry). Carry flag is also set on 32bit unsigned overflow and can be read with the CY() function.
Description	Performs an unsigned subtraction of 2 x 32bit values placing the 32bit result in a 2 word array.	
Example	<pre>var carry, valA[2], valB[2], Result[2]; var p; valA[0] := 0; valA[1] := 0xFFFF; valB[0] := 0; valB[1] := 0xEFFF; carry := usub_3232(Result, valA, valB); p := str_Ptr(Result); print("0x"); str_Printf(&p, "%lX"); repeat forever</pre> <p>This example will print 0x10000000</p>	

2.3.17. umul_1616(&res32, val1, val2)

Syntax	umul_1616(&res32, val1, val2);	
Arguments	&res32, val1, val2	
	&res32	Points to 32bit result register.
	val1	16bit register or constant
	val2	16bit register or constant
Returns	Pointer	
	Pointer	Returns a pointer to the 32bit result. Carry and overflow are not affected.
Description	Performs an unsigned multiply of 2 x 16bit values placing the 32bit result in a 2 word array.	
Example	<pre>var val32[2]; var p; umul_1616(val32, 500, 2000); p := str_Ptr(val32); str_Printf(&p, "%ld");</pre> <p>This example prints 1000000</p>	

2.3.18. udiv_3232(&res32, val1, val2)

Syntax	udiv_3232(&res32, val1, val2);	
Arguments	&res32, val1, val2	
	&res32	Points to 32bit result register.
	val1	32bit register or dividend
	val2	32bit register or divisor
Returns	Pointer	
	Pointer	Returns a pointer to the 32bit result. Carry and overflow are not affected.
Description	Performs an unsigned division of 2 x 32bit values placing the 32bit result in a 2 word array. Note: A division by zero will result is 0xFFFFFFFF	
Example	<pre> var val32[2], dividend[2], divisor[2] ; var p; dividend[0] := 0x5c21 ; // part of 1661985 dividend[1] := 0x19 ; // part of 1661985 divisor[0] := 13 ; divisor[1] := 0 ; udiv_3232(val32, dividend, divisor); p := str_Ptr(val32); str_Printf(&p, "%ld"); // 1661985 / 13 = 127845 </pre>	

2.3.19. `ucmp_3232(&val1, &val2)`

Syntax	<code>ucmp_3232(&val1, &val2);</code>	
Arguments	<code>&val1, &val2</code>	
	<code>&val1</code>	points to 32bit constant
	<code>&val2</code>	points to 32bit constant
Returns	value	
	value	0 if equal 1 if <code>val1 > val2</code> -1 if <code>val1 < val2</code> This function does not affect the carry flag.
Description	Performs an unsigned comparison of 2 x 32bit values.	
Example	<pre> var carry, valA[2], valB[2], Result; valA[0] := 0; valA[1] := 0xFFFF; valB[0] := 0; valB[1] := 0xEFFF; Result := cmp_3232(valA, valB); //val1 > val2 print(Result); repeat forever </pre> <p>This example will print 1.</p>	

2.4. Text and String Functions

Summary of Functions in this section:

- `txt_MoveCursor(line, column)`
- `putch(char)`
- `putchXY(xpos, ypos, char)`
- `putstr(pointer)`
- `putstrXY(xpos, ypos, string)`
- `putstrCentred(xc, yc, string)`
- `putnum(format, value)`
- `print(...)`
- `to(outstream)`
- `charwidth(char)`
- `charheight(char)`
- `strwidth(pointer)`
- `strheight()`
- `strlen(pointer)``txt_Set(function, value)`
- `unicode_page(charbeg, charend, charoffset)`
- `txt_Set(function, value)`
txt_Set shortcuts:
 - `txt_FGcolour(colour)`
 - `txt_BGcolour(colour)`
 - `txt_FontID(id)`
 - `txt_Width(multiplier)`
 - `txt_Height(multiplier)`
 - `txt_Xgap(pixelcount)`
 - `txt_Ygap(pixelcount)`
 - `txt_Delay(millisecs)`
 - `txt_Opacity(mode)`
 - `txt_Bold(mode)`
 - `txt_Italic(mode)`
 - `txt_Inverse(mode)`
 - `txt_Underline(mode)`
 - `txt_Attributes(value)`
 - `txt_Wrap (value)`
 - `txt_Angle(value)`
- `txt_FontBank(bank, address)`
- `PutnumXY(x, y, format, value)`

2.4.1. txt_MoveCursor(line, column)

Syntax	<code>txt_MoveCursor(line, column);</code>	
Arguments	line, column	
	line	Holds a positive value for the required line position.
	column	Holds a positive value for the required column position.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Moves the text cursor to a screen position set by line and column parameters. The line and column position is calculated, based on the size and scaling factor for the currently selected font. When text is outputted to screen it will be displayed from this position. The text position could also be set with <code>gfx_MoveTo(...)</code> ; if required to set the text position to an exact pixel location. Note that lines and columns start from 0. So, line 0, column 0 is the top left corner of the display.	
Example	<pre>txt_MoveCursor(4, 9);</pre> <p>This example moves the text origin to the 5th line and the 10th column.</p>	

2.4.2. `putch(char)`

Syntax	<code>putch(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Holds a positive value for the required character.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	<code>putch</code> prints single characters to the current output stream, usually the display.	
Example	<pre>var v; v := 0x39; putch(v); // print the number 9 to the current display location putch('\n'); // newline</pre>	

2.4.3. `putchXY(xpos, ypos, char)`

Syntax	<code>putchXY(xpos, ypos, char);</code>	
Arguments	<code>xpos, ypos, char</code>	
	<code>xpos</code>	Specifies the horizontal position of the character.
	<code>ypos</code>	Specifies the vertical position of the character.
	<code>char</code>	Holds a positive value for the required character.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	<code>putch</code> prints a single character at position <code>x, y</code> .	
Example	<pre>var v; v := 0x39; putch(10, 20, v); // print the number 9 to x,y (10,20) putch('\n'); // newline</pre>	

2.4.4. putstr(pointer)

Syntax	<code>putstr(string);</code>	
Arguments	<code>string</code>	
	<code>string</code>	A string constant, a pointer to a string, a pointer to an array, or a pointer to a data statement.
Returns	<code>source</code>	
	<code>source</code>	Returns the pointer to the item that was printed.
Description	<p><code>putstr</code> prints a string to the current output stream, usually the display.</p> <p>Note: The string constants and data statement pointers are byte aligned.</p> <p>Note: <code>putstr</code> is more efficient than <code>print</code> for printing single strings.</p> <p>Note: The output of <code>putstr</code> can be redirected to the communications port, the media, or memory using the <code>to(...);</code> function.</p> <p>A string constant is automatically terminated with a zero.</p> <p>A string in a data statement is not automatically terminated with a zero.</p> <p>All variables in 4DGL are 16bit, if an array is used for holding 8 bit characters; each array element packs 1 or 2 characters.</p>	
Example	<pre>//===== // Example #1 - print a string constant //===== putstr("HELLO\n"); //simply print a string constant at current origin //===== // Example #2 - print string via pointer //===== var p; // a var for use as a pointer p := "String Constant\n"; // assign a string constant to pointer s putstr(p); // print the string using the pointer putstr(p+8); // print, offsetting into the string //===== // Example #3 - printing strings from data table //===== #DATA byte message "Week",0 word days sun,mon,tue,wed,thu,fri,sat // pointers to data items byte sun "Sunday\n\0" byte mon "Monday\n\0" byte tue "Tuesday\n\0" byte wed "Wednesday\n\0" byte thu "Thursday\n\0" byte fri "Friday\n\0" byte sat "Saturday\n\0" #END var n; putstr n:=0; while(n < 7)</pre>	

```
putstr(days[n++]); // print the days  
wend
```

2.4.5. putstrXY(xpos, ypos, string)

Syntax	<code>putstr(xpos, ypos, string);</code>
Arguments	<code>xpos, ypos, string</code>
	xpos Specifies the horizontal position of the string.
	ypos Specifies the vertical position of the string.
	string A string constant, a pointer to a string, a pointer to an array, or a pointer to a data statement.
Returns	nothing
Description	<p><code>putstr</code> prints a string at position x, y on the display.</p> <p>Note: The string constants and data statement pointers are byte aligned.</p> <p>A string constant is automatically terminated with a zero.</p> <p>A string in a data statement is not automatically terminated with a zero.</p> <p>All variables in 4DGL are 16bit, if an array is used for holding 8 bit characters; each array element packs 1 or 2 characters.</p>
Example	<pre>//===== // Example #1 - print a string constant //===== putstrXY(5,10, "HELLO\n"); //Print 'Hello' at 5,10 //===== // Example #2 - print string via pointer //===== var p; // a var for use as a pointer p := "String Constant\n"; // assign a string constant to pointer s putstr(p); // print the string using the pointer putstr(5, 10, p+8); // print at 5,10, offsetting into the string //===== // Example #3 - printing strings from data table //===== #DATA byte message "Week",0 word days sun,mon,tue,wed,thu,fri,sat // pointers to data items byte sun "Sunday\0" byte mon "Monday\0" byte tue "Tuesday\0" byte wed "Wednesday\0" byte thu "Thursday\0" byte fri "Friday\0" byte sat "Saturday\0" #END var n; n:=0; while(n < 7) putstrXY(0, n+10, days[n++]); // print the days wend</pre>

2.4.6. `putstrCentred(xc, yc, string)`

Syntax	<code>putstr(xc, yc, string);</code>	
Arguments	<code>xc, yc, string</code>	
	<code>xc</code>	Specifies the horizontal position of the string.
	<code>yc</code>	Specifies the vertical position of the string.
	<code>string</code>	A string constant, a pointer to a string, a pointer to an array, or a pointer to a data statement.
Returns	nothing	
Description	<p><code>putstr</code> prints a string centered at position x, y on the display.</p> <p>Note: The string constants and data statement pointers are byte aligned.</p> <p>A string constant is automatically terminated with a zero.</p> <p>A string in a data statement is not automatically terminated with a zero.</p> <p>All variables in 4DGL are 16bit, if an array is used for holding 8 bit characters; each array element packs 1 or 2 characters.</p>	
Example	<pre>putstrCentred(120, 0, "4D Systems\n"); //Print 'Hello' at 5,10</pre> <p>Assuming X-resolution = 240, this command will print '4D systems' in the top-middle of the screen.</p>	

									BIN12	BIN12Z	BIN12ZB
									BIN13	BIN13Z	BIN13ZB
									BIN14	BIN14Z	BIN14ZB
									BIN15	BIN15Z	BIN15ZB
									BIN16	BIN16Z	BIN16ZB
Returns											
field											
	field	Returns the the default width of the numeric field (digit count), usually ignored.									
Description											
	putnum	prints a 16bit number in various formats to the current output stream, usually the display.									
Example											
	<pre>var v; v := 05678; putnum(HEX, v); // print the number as hex 4 digits putnum(BIN, v); // print the number as binary 16 digits</pre>										

2.4.8. print(...)

Syntax	<code>print(...);</code>
Arguments	See Description
Returns	nothing
Description	<p>4DGL has a versatile <code>print(...)</code> statement for formatting numbers and strings. In it's simplest form, print will simply print a number as can be seen below:</p> <pre>myvar := 100; print(myvar);</pre> <p>This will print 100 to the current output device (usually the display in TEXT mode). Note that if you wish to add a string anywhere within a <code>print(...)</code> statement, just place a quoted string expression and you will be able to mix strings and numbers in a variety of formats. See the following example.</p> <pre>print("the value of myvar is :- ", myvar, "and its 8bit binary representation is:-", [BIN8]myvar);</pre> <p>* Refer the the table in putnum(..) for all the numeric representations available.</p> <p>The <code>print(...)</code> statement will accept directives passed in square brackets to make it print in various ways, for instance, if you wish to print a number in 4 digit hex, use the [HEX4] directive placed in front of the variable to be displayed within the print statement. See the following example.</p> <pre>print("myvar as a 4 digit HEX number is :- ", [HEX4]myvar);</pre> <p>Note that there are 2 print directives that are not part of the numeric set and will be explained separately. these are the [STR] and [CHR] directives.</p> <p>The [STR] directive expects a string pointer to follow:</p> <pre>s := "Hello World"; // assign a string constant to s print("Var 's' points to a string constant at address", s, " which is", [STR] s);</pre> <p>The [CHR] directive prints the character value of a variable.</p> <pre>print("The third character of the string is '", [CHR] *(s+2));</pre> <p>also</p> <pre>print("The value of 'myvar' as an ASCII charater is '", [CHR] myvar);</pre> <p>Note that you can freely mix string pointers, strings, variables and expressions within a print statement. <code>print(...)</code> can also use the <code>to(...)</code> function to redirect it's output to a different output device other than the screen using the function (refer to the to(...) statement for further examples).</p>
Example	<pre>#platform "uLCD-70DT" //////////////////// // DATA STATEMENT // //////////////////// #DATA</pre>

```
word myData
myString1, Bert, Fred, main, myString2, baud, barney,
0x1111,0x2222,0x3333,0x4444

byte myString1 "Data String OK\n\n",0
byte myString2 "\"(and forward referenced!)\n\n",0
word baud 150,300,600,1200,2400,9600
#END

// this constant is a forward reference
#constant barney 9876

func Fred(var str)
print("string = ", [STR] str);
endfunc

func Bert(var p1, var p2, var p3)
print("hello from Bert\np1=",p1,"np2=",p2, "\np3=",p3,"\n");
return "Bert was here\n";
endfunc

func main()
var fn; // a variable for a handle for the function

txt_Set(FONT_ID, FONT_1);

fn := myData[1]; //Get function pointer from data statement index
print( [STR] fn(100,200,300) );
// use it in a statement to prove engine ok

fn := myData[2]; //Get function pointer from data statement index
fn("ABC\n"); // execute the function

// just shows where main lives
print("\naddress of main = code[", myData[3],"]\n\n");
// remember - a var can be a handle, variable, pointer or vector
print( [STR] myData[0]); // pointer table data reference
print( [STR] myData[4]);

repeat forever

endfunc
```

2.4.9. to(outstream)

Syntax	to(outstream);		
Arguments	outstream		
	outstream	A variable or constant specifying the destination for the putch , putstr , putnum , print and str_Printf functions.	
	Predefined Name	Constant	putch() , putstr() , putnum() , print() , str_Printf() redirection
	DSK	0xF802	Output is directed to the most recently open file that has been opened in write mode.
	COM0	0xF804	Output is redirected to the COM0 (default serial) port.
	COM1	0xFF05	Output is redirected to the COM1 (default serial) port.
	COM2	0xFF06	Output is redirected to the COM2 (default serial) port.
	COM3	0xFF07	Output is redirected to the COM3 (default serial) port.
	I2C1	0xF820	Output is directed to the I2C1 port.
	I2C2	0xF821	Output is directed to the I2C2 port.
	I2C3	0xF822	Output is directed to the I2C3 port.
	MDA	0xF840	Output is directed to the SD/SDHC or FLASH media. Warning – be careful writing to a FAT16 formatted card without checking legal partitioned are else the disk formatting will be destroyed.
	APPEND	0x0000	Output is appended to user array if previous redirection was to an array.
	(memory pointer)	Array address	Output is redirect to the memory pointer argument.
Returns	nothing		
Description	<p>to() sends the printed output to destinations other than the screen. Normally, print just sends its output to the display in TEXT mode which is the default, however, the output from print can be sent to 'streams', eg – COM0, COM1, COM2, or COM3, an open FAT16 file with DSK, to raw media with MDA (media), or to the I2C ports with I2C1, I2C2 or I2C3.</p> <p>The to(...) function can also stream to a memory array . Note that once the to(...) function has taken effect, the stream reverts back to the default stream which is TEXT as soon as putch, putstr, putnum, print, or str_Printf has completed its action.</p> <p>The APPEND argument is used to append the printed output to the same place as the previous redirection. This is most useful for building string arrays, or adding sequential data to a media stream.</p>		
Example	<pre>//===== // Example #1 - putstr redirection //===== var buf[10]; // a buffer that will hold up to 20 bytes/chars var s; // a var for use as a pointer</pre>		

```
to(buf); putstr("ONE "); // redirect putstr to the buffer
to(APPEND); putstr("TWO "); // and add a couple more items
to(APPEND); putstr("THREE\n");
putstr(buf); // print the result to the display

while (media_Init()==0); // wait if no SD/SDHC card detected
media_SetSector(0, 2); // at sector 2
//media_SetAdd(0, 1024); // (alternatively, use media_SetAdd(),
// lower 9 bits ignored).
to(MDA); putstr("Hello World"); // now write a ascii test string
media_WriteByte('A'); // write a further 3 bytes
media_WriteByte('B');
media_WriteByte('C');
to(MDA); putstr(buf); // write the buffer we prepared earlier
media_WriteByte(0); // terminate with ASCII zero
media_Flush();
media_SetAdd(0, 1024); // reset the media address
while(char:=media_ReadByte())
    to(COM0); putch(char); // print the stored string to the COM port
wend
repeat forever
```

2.4.10. charwidth('char')

Syntax	<code>charwidth('char');</code>	
Arguments	<code>'char'</code>	
	<code>'char'</code>	The ascii character for the width calculation.
Returns	<code>width</code>	
	<code>width</code>	Returns the width of a single character in pixel units.
Description	<code>charwidth</code> is used to calculate the width in pixel units for a character, based on the currently selected font.	
Example	<pre>//===== // Example //===== str := "HELLO\nTHERE"; // note that this string spans 2 lines due // to the \n. width := strwidth(str); // get the width of the string, this will // also capture the height. height := strheight(); // note, invoking strwidth also calcs height // which we can now read. // The string above spans 2 lines, strheight(.) will calculate height // correctly for multiple lines. len := strlen(str); // the strlen() function returns the number // of characters in a string. print("\nLength=",len); // NB:- the \n in "HELLO\nTHERE" is counted // as a character. txt_FontID(MS_SanSerif8x12); // select this font w := charwidth('W'); // get a characters width h := charheight('W'); // and height txt_FontID(0); // back to default font print ("\n'W' is " ,w, " pixels wide"); // show width of a character // 'W' in pixel units. print ("\n'W' is " ,h, " pixels high"); // show height of a character // 'W' in pixel units.</pre>	

2.4.11. charheight('char')

Syntax	<code>charheight('char');</code>	
Arguments	<code>'char'</code>	
	<code>'char'</code>	The ascii character for the height calculation.
Returns	<code>width</code>	
	<code>width</code>	Returns the height of a single character in pixel units.
Description	<code>charheight</code> is used to calculate the height in pixel units for a character, based on the currently selected font.	
Example	See <code>example</code> in <code>charwidth()</code>	

2.4.12. **strwidth(pointer)**

Syntax	strwidth(pointer);	
Arguments	pointer	
	pointer	The pointer to a zero (0x00) terminated string.
	'pointer' may be a constant or pointer to word aligned variable.	
Returns	width	
	width	Returns the width of a string in pixel units, can be multi line.
Description	strwidth returns the width of a zero terminated string in pixel units. Note that any string constants declared in your program are automatically terminated with a zero as an end marker by the compiler. Any string that you create in the DATA section or MEM section must have a zero added as a terminator for this function to work correctly.	
Example	See example in charwidth()	

2.4.13. strheight()

Syntax	<code>strheight();</code>	
Arguments	none	
Returns	height	
	height	Returns the height of a string in pixel units, can be multi line.
Description	<p>strheight returns the height of a zero terminated string in pixel units. The strwidth function must be called first which makes available width and height. Note that any string constants declared in your program are automatically terminated with a zero as an end marker by the compiler. Any string that you create in the DATA section or MEM section must have a zero added as a terminator for this function to work correctly.</p>	
Example	See <code>example</code> in <code>charwidth()</code>	

2.4.14. strlen(pointer)

Syntax	strlen(pointer);	
Arguments	pointer	
	pointer	The pointer to a zero (0x00) terminated string.
Returns	length	
	length	Returns the length of a string in character units.
Description	strlen returns the length of a zero terminated string in character units. Note that any string constants declared in your program are automatically terminated with a zero as an end marker by the compiler. Any string that you create in the DATA section or MEM section must have a zero added as a terminator for this function to work correctly.	
Example	See <code>example</code> in <code>charwidth()</code>	

2.4.15. unicode_page(charbeg, charend, charoffset)

Syntax	<code>unicode_page(charbeg, charend, charoffset);</code>	
Arguments	<code>charbeg, charend, charoffset</code>	
	<code>charbeg</code>	Offset of first character in Unicode set.
	<code>charend</code>	Offset of ending character in Unicode Set.
	<code>charoffset</code>	Offset of first ASCII character in Unicode Set.
Returns	<code>count</code>	
	<code>count</code>	Returns count of characters in the set.
Description	<p>After selecting a Unicode image control with <code>txt_FontID</code>, this function is called to set the required font within the Unicode set. The file "Unicode.inc" contains wrappers for this function, and it is not normally called directly.</p> <p>Refer to Unicode documentation <i>'4DGL-Unicode-REVx.pdf'</i> and <i>'Unicode.inc'</i> for further information.</p>	
Example	See Unicode.inc	

2.4.16. txt_Set(function, value)

Syntax	<code>txt_Set(function, value);</code>		
Arguments	<code>function, value</code>		
	function	The function number determines the required action for various text control functions. Usually a constant, but can be a variable, array element, or expression. There are pre-defined constants for each of the functions.	
	value	A variable, array element, expression or constant holding a value for the selected function.	
Returns	nothing		
Description	Given a function number and a value, set the required text control parameter, such as size, colour, and other formatting controls. This function is extremely useful in a loop to select multiple parameters from a data statement or a control array. Note also that each function available for txt_Set has a single parameter 'shortcut' function that has the same effect. (see the Single parameter short-cuts for the txt_Set functions next page)		
	function		value
#	Predefined Name	Description	
0	TEXT_COLOUR	Set the text foreground colour	Colour 0-65535 Default = LIME
1	TEXT_HIGHLIGHT	Set the text background colour	Colour 0-65535 Default = BLACK
2	FONT_ID	Set the required font. System_5x7 System_8x8 System_8x12 System_12x16 MS_SanSerif8x12 dejaVuSans9pt dejaVuSansBold9pt dejaVuSansCondensed9pt System_3x6 plotted EGA 8x12 font Note: The value could be the name of a custom font included in a users program in a data statement.	1 or FONT_1 2 or FONT_2 3 or FONT_3 4 or FONT_4 5 or FONT_5 6 or FONT_6 7 or FONT_7 8 or FONT_8 9 or FONT_9 10 or FONT_10 11 or FONT_11 Default = FONT_3
3	TEXT_WIDTH	Set the text width multiplier. Text will be printed magnified horizontally by this factor	1 to 16 Default = 1
4	TEXT_HEIGHT	Set the text height multiplier. Text will be printed magnified vertically by this factor.	1 to 16 Default = 1
5	TEXT_XGAP	Set the pixel gap between characters. The gap is in pixel units	0 to 32 Default = 0
6	TEXT_YGAP	Set the pixel gap between lines. The gap is in pixel units.	0 to 32 Default = 0
7	TEXT_PRINTDELAY	Set the delay between character printing to give a 'teletype' like effect.	0 to 255 Default = 0msec

8	TEXT_OPACITY	Selects whether or not the 'background' pixels are drawn (default mode is OPAQUE)	0 or TRANSPARENT 1 or OPAQUE Default = 0
9	TEXT_BOLD	Sets Bold Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
10	TEXT_ITALIC	Sets Italic Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
11	TEXT_INVERSE	Sets Inverse Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
12	TEXT_UNDERLINED	Sets Underlined Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
13	TEXT_ATTRIBUTES	Allows a combination of text attributes to be defined together by 'or'ing the bits together. The feature automatically resets after printing using putstr or print has completed. Example: txt_Set(TEXT_ATTRIBUTES, BOLD INVERSE); // bold + inverse Note: bits 0-3 and 8-15 are reserved	16 or BOLD 32 or ITALIC 64 or INVERSE 128 or UNDERLINED
14	TEXT_WRAP	Sets the pixel position where text wrap will occur at RHS The feature automatically resets when screen mode is changed. If the value is set to 0, text wrap is turned off of the current screen. Note: The value is in pixel units.	0 to n(OFF or Value) Default = 0
15	TEXT_ANGLE	Sets the text angle, only for plotted fonts. The feature automatically resets when screen mode is changed.	0 to 359 degrees

Single parameter short-cuts for the txt_Set(..) functions

Function Syntax	Function Action	value
txt_FGcolour(colour)	Set the text foreground colour	Colour 0-65535 Default = LIME
txt_BGcolour(colour)	Set the text background colour	Colour 0-65535 Default = BLACK
txt_FontID(id)	Set the required font. System_5x7 System_8x8 System_8x12 System_12x16 MS_SanSerif8x12 dejaVuSans9pt dejaVuSansBold9pt dejaVuSansCondensed9pt System_3x6 plotted EGA 8x12 font Note: The value could also be the name of a custom font	1 or FONT_1 2 or FONT_2 3 or FONT_3 4 or FONT_4 5 or FONT_5 6 or FONT_6 7 or FONT_7 8 or FONT_8 9 or FONT_9 10 or FONT_10 11 or FONT_11 Default = FONT_3

	included in a users program in a data statement, or the handle returned from file_LoadImageControl() for a uSD based font.	
txt_Width(multiplier)	Set the text width multiplier. Text will be printed magnified horizontally by this factor	1 to 16 Default = 1
txt_Height(multiplier)	Set the text height multiplier. Text will be printed magnified vertically by this factor.	1 to 16 Default = 1
txt_Xgap(pixelcount)	Set the pixel gap between characters. The gap is in pixel units	0 to 32 Default = 0
txt_Ygap(pixelcount)	Set the pixel gap between lines. The gap is in pixel units.	0 to 32 Default = 0
txt_Delay(millisecs)	Set the delay between character printing to give a 'teletype' like effect.	0 to 255 Default = 0msec
txt_Opacity(mode)	Selects whether or not the 'background' pixels are drawn (default mode is OPAQUE)	0 or TRANSPARENT 1 or OPAQUE Default = 0
txt_Bold(mode)	Sets Bold Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
txt_Italic(mode)	Sets Italic Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
txt_Inverse(mode)	Sets Inverse Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
txt_Underline(mode)	Sets Underline Text mode for the next string or char. The feature automatically resets after printing using putstr or print has completed.	0 or 1 (ON or OFF)
txt_Attributes(value)	Allows a combination of text attributes to be defined together by 'or'ing the bits together. The feature automatically resets after printing using putstr or print has completed. Example: <code>txt_Set(TEXT_ATTRIBUTES, BOLD INVERSE); // bold + inverse</code> Note: bits 0-3 and 8-15 are reserved	16 or BOLD 32 or ITALIC 64 or INVERSE 128 or UNDERLINED
txt_Wrap(value)	Sets the pixel position where text wrap will occur at RHS The feature automatically resets when screen mode is changed. If the value is set to 0, text wrap is turned off of the current screen. Note: The value is in pixel units.	0 to n(OFF or Value) Default = 0
txt_Angle(value)	Sets the text angle, only for plotted fonts. The feature automatically resets when screen mode is changed.	0 to 359 degrees

2.4.17. txt_FontBank(bank, address)

Syntax	<code>txt_FontBank(bank, address);</code>	
Arguments	<code>bank, address</code>	
	<code>bank</code>	The bank that the font is stored in
	<code>address</code>	The address of the font within the bank
Returns	<code>font</code>	
	<code>font</code>	Returns the current font before the change.
Description	Enables the usage of fonts stored in banks. See the <code>FontInBankTest</code> and <code>BookAntiqua2032FontsInBank1</code> samples. If a single font is the only thing in a bank its address will be 7, otherwise look in the <code>.lst</code> file from the compile to find the address of the font. Assuming there is space available multiple fonts can be stored in the same bank.	
Example	<code>txt_FontBank(FONTBANK_1, 7) ;</code>	

2.4.18. PutnumXY(x, y, format, value)

Syntax	<code>putnumXY(x, y, format, value);</code>	
Arguments	<code>x, y, format, value</code>	
	x	The x position to start printing the number in.
	y	The y position to start printing the number in.
	format	A constant that specifies the number format.
	value	The number to be printed.
Returns	field	
	field	Returns the the default width of the numeric field (digit count), usually ignored.
Description	<code>putnumXY</code> prints a 16bit number in various formats to the current output stream, usually the display at the specified position. The Formats are the same as for the <code>putnum</code> command	
Example	<pre>var v; v := 05678; putnumXY(0, 0, HEX, v); // print the number as hex 4 digits putnumXY(0, 20, BIN, v); // print the number as binary 16 digits</pre>	

2.5. Ctype Functions

Summary of Functions in this section:

- isdigit(char)
- isxdigit(char)
- isupper(char)
- islower(char)
- isalpha(char)
- isalnum(char)
- isprint(char)
- isspace(char)
- iswhite(char)
- toupper(char)
- tolower(char)
- LObyte(var)
- HIbyte(var)
- ByteSwap(var)
- NybleSwap(var)

2.5.1. isdigit(char)

Syntax	isdigit(char);	
Arguments	char	
	char	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not as ASCII digit 1: Character is an ASCII digit.
Description	Tests the character parameter and returns a 1 if the character is an ASCII digit else returns a 0. Valid range: "0123456789".	
Example	<pre>func main() var ch; var stat; gfx_Cls(); txt_Set(FONT_ID, FONT_2); print ("Serial Input Test\n"); print ("Download prog to flash\n"); print ("Then use debug terminal\n"); to(COM0); print("serial input test:\n"); // now just stay in a loop repeat ch := serin(); if (ch != -1) print([CHR] ch); // if a key was received from PC, // print its ascii value if (isdigit(ch)) print("Character is an ASCII digit"); if (isxdigit(ch)) print("Character is ASCII Hexadecimal"); if (isupper(ch)) print("Character is ASCII uppercase letter"); if (islower(ch)) print("Character is ASCII lowercase letter"); if (isalpha(ch)) print("Character is an ASCII uppercase or lowercase"); if (isalnum(ch)) print("Character is an ASCII Alphanumeric"); if (isprint(ch)) print("Character is a printable ASCII"); if (isspace(ch)) print("Character is a space type character"); endif forever endfunc;</pre>	

2.5.2. isxdigit(char)

Syntax	isxdigit(char);	
Arguments	char	
	char	Specifies the ASCII character for the test.
Returns	Status	
	Status	0 : Character is not as ASCII hexadecimal digit 1 : Character is an ASCII hexadecimal digit.
Description	Tests the character parameter and returns a 1 if the character is an ASCII hexadecimal digit else returns a 0. Valid range: "0123456789ABCDEF".	
Example	Refer to Sec 2.5.1	

2.5.3. isupper(char)

Syntax	<code>isupper(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not an ASCII upper case letter. 1: Character is an ASCII upper case letter.
Description	Tests the character parameter and returns a 1 if the character is an ASCII upper case letter else returns a 0. Valid range: "ABCDEF...WXYZ".	
Example	Refer to Sec 2.5.1	

2.5.4. islower(char)

Syntax	<code>islower(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not an ASCII lower case letter 1: Character is an ASCII lower case letter.
Description	Tests the character parameter and returns a 1 if the character is an ASCII lower case letter else returns a 0. Valid range: "abcd...wxyz".	
Example	Refer to Sec 2.5.1	

2.5.5. isalpha(char)

Syntax	isalpha(char);	
Arguments	char	
	char	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not as ASCII lower or upper case letter. 1: Character is an ASCII lower or upper case letter..
Description	Tests the character parameter and returns a 1 if the character is an ASCII lower or upper case letter else returns a 0. Valid range : "abcd...wxyz", "ABCD...WXYZ"	
Example	Refer to Sec 2.5.1	

2.5.6. isalnum(char)

Syntax	isalnum(char);	
Arguments	char	
	char	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not as ASCII Alphanumeric character. 1: Character is an ASCII Alphanumeric character.
Description	Tests the character parameter and returns a 1 if the character is an ASCII Alphanumeric else returns a 0. Valid range : "abcd...wxyz", "ABCD...WXYZ", "0123456789"	
Example	Refer to Sec 2.5.1	

2.5.7. isprint(char)

Syntax	<code>isprint(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not a printable ASCII character. 1: Character is a printable ASCII character.
Description	Tests the character parameter and returns a 1 if the character is a printable ASCII character else returns a 0. Valid range : 0x20... 0x7F	
Example	Refer to Sec 2.5.1	

2.5.8. isspace(char)

Syntax	<code>isspace(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the ASCII character for the test.
Returns	Status	
	Status	0: Character is not a space type character. 1: Character is a space type character.
Description	Tests the character parameter and returns a 1 if the character is any one of the space type character else returns a 0. Valid range : space, formfeed, newline, carriage return, tab, vertical tab.	
Example	Refer to Sec 2.5.1	

2.5.9. toupper(char)

Syntax	<code>toupper(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the ASCII character for the test.
Returns	<code>char</code>	
	<code>char</code>	"ABCD...WXYZ" : If character is lower case letter. char : If character is not a lower case letter.
Description	Tests the character parameter and if the character is a lower cases letter, it returns the upper case equivalent else returns the passed char. Valid range: "abcd ... wxyz".	
Example	<pre> func main() var ch, Upconvch, Loconvch; var stat; gfx_Cls(); txt_Set(FONT_ID, FONT2); print ("Serial Input Test\n"); print ("Download prog to flash\n"); print ("Then use debug terminal\n"); to(COM0); print("serial input test:\n"); // now just stay in a loop repeat ch := serin(); if (ch != -1) print([CHR] ch); // if a key was received from PC, // print its ascii value if (isupper(ch)) print("Uppercase ASCII found. Converting to lowercase"); Loconvch := tolower(ch); endif if (islower(ch)) print("Lowercase ASCII found. Converting to Uppercase"); Upconvch := toupper(ch); endif endif forever endfunc; </pre>	

2.5.10. tolower(char)

Syntax	<code>tolower(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the ASCII character for the test.
Returns	Status	
	Status	"abcd...wxyz" : If character is upper case letter. char : If character is not a upper case letter...
Description	Tests the character parameter and if the character is a lower case letter it returns the upper case equivalent else returns the passed char. Valid range: "ABCD ... WXYZ".	
Example	Refer to Sec 2.5.9	

2.5.11. LByte(var)

Syntax	LByte(var);	
Arguments	var	
	var	User variable.
Returns	byte	
	byte	Returns the lower byte (lower 8 bit) of a 16 bit variable.
Description	Returns the lower byte (lower 8 bit) of a 16 bit variable.	
Example	<code>myvar := LByte(myvar2);</code>	

2.5.12. HIbyte(var)

Syntax	HIbyte(var);	
Arguments	var	
	var	User variable.
Returns	byte	
	byte	Returns the upper byte (upper 8 bits) of a 16 bit variable.
Description	Returns the upper byte (upper 8 bits) of a 16 bit variable.	
Example	<code>myvar := HIbyte(myvar2);</code>	

2.5.13. ByteSwap(var)

Syntax	ByteSwap(var);	
Arguments	var	
	var	User variable.
Returns	value	
	value	Returns the endian swapped value of a 16 bit variable.
Description	Returns the swapped upper and lower bytes of a 16 bit variable.	
Example	<code>myvar := ByteSwap(myvar2);</code>	

2.5.14. NybleSwap(var)

Syntax	NybleSwap(var);	
Arguments	var	
	var	User variable.
Returns	value	
	value	Returns the 16 bit variable with swapped lower nybles
Description	Returns the swapped lower bytes nybles, upper byte retained	
Example	<code>myvar := NybleSwap(myvar2);</code>	

2.6. Graphics Functions

Summary of Functions in this section:

- gfx_Cls()
- gfx_ChangeColour(oldColour, newColour)
- gfx_Circle(x, y, radius, colour)
- gfx_CircleFilled(x, y, radius, colour)
- gfx_Line(x1, y1, x2, y2, colour)
- gfx_Hline(y, x1, x2, colour)
- gfx_Vline(x, y1, y2, colour)
- gfx_Rectangle(x1, y1, x2, y2, colour)
- gfx_RectangleFilled(x1, y1, x2, y2, colour)
- gfx_RoundRect(x1, y1, x2, y2, rad, colour)
- gfx_Polyline(n, vx, vy, colour)
- gfx_Polygon(n, vx, vy, colour)
- gfx_Triangle(x1, y1, x2, y2, x3, y3, colour)
- gfx_Dot()
- gfx_Bullet(radius)
- gfx_OrbitInit(&x_dest, &y_dest)
- gfx_Orbit(angle, distance)
- gfx_PutPixel(x, y, colour)
- gfx_GetPixel(x, y)
- gfx_MoveTo(xpos, ypos)
- gfx_MoveRel(xoffset, yoffset)
- gfx_IncX()
- gfx_IncY()
- gfx_LineTo(xpos, ypos)
- gfx_LineRel(xpos, ypos)
- gfx_BoxTo(x2, y2)
- gfx_SetClipRegion()
- gfx_Ellipse(x, y, xrad, yrad, colour)
- gfx_EllipseFilled(x, y, xrad, yrad, colour)
- gfx_Button(state, x, y, buttonColour, textColour, font, textWidth, textHeight, text)
- gfx_Button2(state, x, y, width, height, buttonColour, txtColour, text)
- gfx_Button3(state, x, y, width, height, buttonColour, txtColour, text)
- gfx_Panel(state, x, y, width, height, colour)
- gfx_RoundPanel(states, x, y, width, height, radius, bevelwidth, colour)
- gfx_Slider(mode, x1, y1, x2, y2, colour, scale, value)
- gfx_Slider2(mode, x1, y1, width, height, colour, scale, value)
- gfx_ScreenCopyPaste(xs, ys, xd, yd, width, height)
- gfx_RGBto565(RED, GREEN, BLUE)
- gfx_332to565(COLOUR8BIT)
- gfx_565to332(COLOUR)
- gfx_TriangleFilled(x1, y1, x2, y2, x3, y3, colr)
- gfx_PolygonFilled(n, &vx, &vy, colr)
- gfx-Origin(x, y)
- gfx_Get(mode)
- gfx_ClipWindow(x1, y1, x2, y2)
- gfx_Set(function, value)
 - **gfx_Set shortcuts:**
 - gfx_PenSize(mode)
 - gfx_BGcolour(colour)
 - gfx_ObjectColour(colour)
 - gfx_Clipping(mode)

- gfx_TransparentColour(colour)
- gfx_Transparency(mode)
- gfx_FrameDelay(delay)
- gfx_ScreenMode(orientation)
- gfx_OutlineColour(colour)
- gfx_Contrast(value)
- gfx_LinePattern(pattern)
- gfx_BevelRadius(radius)
- gfx_BevelWidth(mode)
- gfx_BevelShadow(value)
- gfx_Xorigin(offset)
- gfx_Yorigin(offset)
- gfx_Arc(xc, radius, step, startangle, endangle, mode)
- gfx_CheckBox(state, x, y, width, height, boxColour, textColour, text)
- gfx_RadioButton(state, x, y, width, height, boxColour, textColour, text)
- gfx_FillPattern(patptr, mode)
- gfx_Gradient(style, x1, y1, x2, y2, colour1, colour2)
- gfx_RoundGradient(style, x1, y1, x2, y2, radius, colour1, colour2)
- gfx_PieSlice(cx, cy, spread, radius, step, startangle, endangle, mode)
- gfx_PointWithinBox(x, y, &rect)
- gfx_PointWithinRectangle(x, y, &recta)
- gfx_ReadBresLine(x1, y1, x2, y2, ptr)
- gfx_WriteBresLine(x1, y1, x2, y2, ptr)
- gfx_ReadGRAMarea(x1, y1, x1, y2, ptr)
- gfx_WriteGRAMarea(x1, y1, x2, y2, ptr)
- gfx_Surround(x1, y1, x2, y2, rad1, rad2, oct, colour)
- gfx_Scope(Left, Width, Yzero, n, Xstep, Yamp, Colourbg, &old_y1, &new_y1, Colour1, ... &old_y4, &new_y4, Colour4)

2.6.1. gfx_Cls()

Syntax	<code>gfx_Cls();</code>
Arguments	none
Returns	nothing
Description	<p>Clear the screen using the current background colour. <code>gfx_Cls()</code> command brings some of the settings back to default; such as,</p> <ul style="list-style-type: none"> • Transparency turned OFF • Outline colour set to BLACK • Opacity set to OPAQUE • Pen set to OUTLINE • Line patterns set to OFF • Right text margin set to full width • Text magnifications set to 1 • All origins set to 0:0 <p>The alternative to maintain settings and clear screen is to draw a filled rectangle with the required background colour.</p>
Example	<pre>gfx_BGcolour (DARKGRAY) ; gfx_Cls () ;</pre> <p>This example clears the entire display using colour DARKGRAY</p>

2.6.2. gfx_ChangeColour(oldColour, newColour)

Syntax	<code>gfx_ChangeColour(oldColour, newColour);</code>	
Arguments	oldColour, newColour	
	oldColour	Specifies the sample colour to be changed within the clipping window.
	newColour	Specifies the new colour to change all occurrences of old colour within the clipping window.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Changes all oldColour pixels to newColour within the clipping area.	
Example	<pre>func main() txt_Width(3); txt_Height(5); gfx_MoveTo(8,20); print("TEST"); // print the string gfx_SetClipRegion(); // force clipping area to extents of text // just printed. gfx_ChangeColour(BLACK, RED); // test change of background colour repeat forever endfunc</pre> <p>This example prints a test string, forces the clipping area to the extent of the text that was printed then changes the background colour.</p>	

2.6.3. gfx_Circle(x, y, radius, colour)

Syntax	<code>gfx_Circle(x, y, rad, colour);</code>	
Arguments	<code>x, y, rad, colour</code>	
	<code>x, y</code>	Specifies the centre of the circle.
	<code>rad</code>	Specifies the radius of the circle.
	<code>colour</code>	Specifies the colour of the circle.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	<p>Draws a circle with centre point x1, y1 with radius r using the specified colour.</p> <p>NB: The default PEN_SIZE is set to OUTLINE, however, if PEN_SIZE is set to SOLID, the circle will be drawn filled, if PEN_SIZE is set to OUTLINE, the circle will be drawn as an outline. If the circle is drawn as SOLID, the outline colour can be specified with <code>gfx_OutlineColour(...)</code>. If OUTLINE_COLOUR is set to 0, no outline is drawn.</p>	
Example	<pre>// assuming PEN_SIZE is OUTLINE gfx_Circle(50,50,30, RED);</pre> <p>This example draws a BLUE circle outline centred at x=50, y=50 with a radius of 30 pixel units.</p>	

2.6.4. gfx_CircleFilled(x, y, radius, colour)

Syntax	<code>gfx_CircleFilled(x, y, rad, colour);</code>	
Arguments	<code>x, y, rad, colour</code>	
	<code>x, y</code>	Specifies the centre of the circle.
	<code>rad</code>	Specifies the radius of the circle.
	<code>colour</code>	Specifies the fill colour of the circle.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	<p>Draws a SOLID circle with centre point x1, y1 with radius using the specified colour.</p> <p>The outline colour can be specified with <code>gfx_OutlineColour(...)</code>. If <code>OUTLINE_COLOUR</code> is set to 0, no outline is drawn.</p> <p>NB:- The <code>PEN_SIZE</code> is ignored, the circle is always drawn SOLID.</p>	
Example	<pre>if(state == TOUCH_RELEASED) // if there's a release; gfx_CircleFilled(x, y, 10, RED); // we'll draw a solid red circle // of radius=10 on touch release endif</pre>	

2.6.5. `gfx_Line(x1, y1, x2, y2, colour)`

Syntax	<code>gfx_Line(x1, y1, x2, y2, colour);</code>	
Arguments	<code>x1, y1, x2, y2, colour</code>	
	<code>x1, y1</code>	Specifies the starting coordinates of the line.
	<code>x2, y2</code>	Specifies the ending coordinates of the line.
	<code>colour</code>	Specifies the colour of the line.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	Draws a line from <code>x1, y1</code> to <code>x2, y2</code> using the specified colour. The line is drawn using the current object colour. The current origin is not altered. The line may be tessellated with the <code>gfx_LinePattern(...)</code> function.	
Example	<pre>gfx_Line(100, 100, 10, 10, RED);</pre> <p>This example draws a RED line from <code>x1=100, y1=100</code> to <code>x2=10, y2=10</code></p>	

2.6.6. gfx_Hline(y, x1, x2, colour)

Syntax	<code>gfx_Hline(y, x1, x2, colour);</code>	
Arguments	<code>y, x1, x2, colour</code>	
	<code>y</code>	Specifies the vertical position of the horizontal line.
	<code>x1, x2</code>	Specifies the horizontal end points of the line.
	<code>colour</code>	Specifies the colour of the horizontal line.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	Draws a fast horizontal line from x1 to x2 at vertical co-ordinate y using colour.	
Example	<pre>gfx_Hline(50, 10, 80, RED);</pre> <p>This example draws a fast RED horizontal line at y=50, from x1=10 to x2=80</p>	

2.6.7. gfx_Vline(x, y1, y2, colour)

Syntax	<code>gfx_Vline(x, y1, y2, colour);</code>	
Arguments	<code>x, y1, y2, colour</code>	
	<code>x</code>	Specifies the horizontal position of the vertical line.
	<code>y1, y2</code>	Specifies the vertical end points of the line.
	<code>colour</code>	Specifies the colour of the vertical line.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	Draws a fast vertical line from y1 to y2 at horizontal co-ordinate x using colour.	
Example	<code>gfx_Vline(20, 30, 70, RED);</code>	
	This example draws a fast RED vertical line at x=20, from y1=30 to y2=70	

2.6.8. gfx_Rectangle(x1, y1, x2, y2, colour)

Syntax	<code>gfx_Rectangle(x1, y1, x2, y2, colour);</code>	
Arguments	<code>x1, y1, x2, y2, colour</code>	
	<code>x1, y1</code>	Specifies the top left corner of the rectangle.
	<code>x2, y2</code>	Specifies the bottom right corner of the rectangle.
	<code>colour</code>	Specifies the colour of the rectangle.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Draws a rectangle from x1, y1 to x2, y2 using the specified colour. The line may be tessellated with the <code>gfx_LinePattern(...)</code> function.</p> <p>NB: The default PEN_SIZE is set to OUTLINE, however, if PEN_SIZE is set to SOLID, the rectangle will be drawn filled, if PEN_SIZE is set to OUTLINE, the rectangle will be drawn as an outline. If the rectangle is drawn as SOLID, the outline colour can be specified with <code>gfx_OutlineColour(...)</code>. If OUTLINE_COLOUR is set to 0, no outline is drawn. The outline may be tessellated with the <code>gfx_LinePattern(...)</code> function.</p>	
Example	<pre>gfx_Rectangle(10, 10, 30, 30, GREEN);</pre> <p>This example draws a GREEN rectangle from x1=10, y1=10 to x2=30, y2=30</p>	

2.6.9. gfx_RectangleFilled(x1, y1, x2, y2, colour)

Syntax	<code>gfx_RectangleFilled(x1, y1, x2, y2, colour);</code>	
Arguments	<code>x1, y1, x2, y2, colour</code>	
	<code>x1, y1</code>	Specifies the top left corner of the rectangle.
	<code>x2, y2</code>	Specifies the bottom right corner of the rectangle.
	<code>colour</code>	Specifies the colour of the rectangle.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	<p>Draws a SOLID rectangle from x1, y1 to x2, y2 using the specified colour. The line may be tessellated with the <code>gfx_LinePattern(...)</code> function.</p> <p>The outline colour can be specified with <code>gfx_OutlineColour(...)</code>. If <code>OUTLINE_COLOUR</code> is set to 0, no outline is drawn. The outline may be tessellated with the <code>gfx_LinePattern(...)</code> function.</p> <p>NB:- The <code>PEN_SIZE</code> is ignored, the rectangle is always drawn SOLID.</p>	
Example	<pre>gfx_RectangleFilled(30,30,80,80, RED);</pre> <p>This example draws a filled RED rectangle from x1=30,y1=30 to x2=80,y2=80</p>	

2.6.10. `gfx_RoundRect(x1, y1, x2, y2, rad, oct, colour)`

Syntax	<code>gfx_RoundRect(x1, y1, x2, y2, rad, colour);</code>	
Arguments	<code>x1, y1, x2, y2, rad, colour</code>	
	<code>x1, y1</code>	Specifies the top left corner of the rectangle.
	<code>x2, y2</code>	Specifies the bottom right corner of the rectangle.
	<code>rad</code>	Specifies the corner radius.
	<code>colour</code>	Specifies the colour of the rectangle.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	<p>Draw a filled rectangle at the given co-ordinates with optional rounded corners. If <code>x1 = x2</code> or <code>y1 = y2</code> no straight line part is drawn.</p> <p>Rendering can be obtained with <code>gfx_FillPattern(PATTRN)</code>; or <code>gfx_FillPattern(OFF)</code>; for no fill pattern determined by 'radius'.</p>	
Example		

2.6.11. `gfx_Polyline(n, vx, vy, colour)`

Syntax	<code>gfx_Polyline(n, vx, vy, colour);</code>	
Arguments	n, vx, vy, colour	
	n	Specifies the number of elements in the x and y arrays specifying the vertices for the polyline.
	vx	Specifies the addresses of the storage of the array of elements for the x coordinates of the vertices.
	vy	Specifies the addresses of the storage of the array of elements for the y coordinates of the vertices.
	colour	Specifies the colour for the lines
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Plots lines between points specified by a pair of arrays using the specified colour. The lines may be tessellated with the <code>gfx_LinePattern(...)</code> function. <code>gfx_Polyline</code> can be used to create complex raster graphics by loading the arrays from serial input or from MEDIA with very little code requirement.</p> <p>This function is very similar to the Ploygon function</p>	
Example	<pre>#inherit "4DGL_16bitColours.fnc" var vx[20], vy[20]; func main() vx[0] := 36; vy[0] := 110; vx[1] := 36; vy[1] := 80; vx[2] := 50; vy[2] := 80; vx[3] := 50; vy[3] := 110; vx[4] := 76; vy[4] := 104; vx[5] := 85; vy[5] := 80; vx[6] := 94; vy[6] := 104; vx[7] := 76; vy[7] := 70; vx[8] := 85; vy[8] := 76; vx[9] := 94; vy[9] := 70; vx[10] := 110; vy[10] := 66; vx[11] := 110; vy[11] := 80; vx[12] := 100; vy[12] := 90; vx[13] := 120; vy[13] := 90; vx[14] := 110; vy[14] := 80; vx[15] := 101; vy[15] := 70; vx[16] := 110; vy[16] := 76; vx[17] := 119; vy[17] := 70; // house gfx_Rectangle(6,50,66,110,RED); // frame gfx_Triangle(6,50,36,9,66,50,YELLOW); // roof gfx_Polyline(4, vx, vy, CYAN); // door</pre>	

```
// man
gfx_Circle(85, 56, 10, BLUE);           // head
gfx_Line(85, 66, 85, 80, BLUE);        // body
gfx_Polyline(3, vx+4, vy+4, CYAN);     // legs
gfx_Polyline(3, vx+7, vy+7, BLUE);     // arms

// woman
gfx_Circle(110, 56, 10, PINK);         // head
gfx_Polyline(5, vx+10, vy+10, BROWN);  // dress
gfx_Line(104, 104, 106, 90, PINK);    // left arm
gfx_Line(112, 90, 116, 104, PINK);    // right arm
gfx_Polyline(3, vx+15, vy+15, SALMON); // dress

repeat forever

endfunc
```

This example draws a simple scene

2.6.12. gfx_Polygon(n, vx, vy, colour)

Syntax	<code>gfx_Polygon(n, vx, vy, colour);</code>	
Arguments	n, vx, vy, colour	
	n	Specifies the number of elements in the x and y arrays specifying the vertices for the polygon.
	vx	Specifies the addresses of the storage of the array of elements for the x coordinates of the vertices.
	vy	Specifies the addresses of the storage of the array of elements for the y coordinates of the vertices.
	colour	Specifies the colour for the polygon
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Plots lines between points specified by a pair of arrays using the specified colour. The last point is drawn back to the first point, completing the polygon. The lines may be tessellated with the <code>gfx_LinePattern(...)</code> function. <code>gfx_Polygon</code> can be used to create complex raster graphics by loading the arrays from serial input or from MEDIA with very little code requirement.	
Example	<pre>var vx[7], vy[7]; func main() vx[0] := 10; vy[0] := 10; vx[1] := 35; vy[1] := 5; vx[2] := 80; vy[2] := 10; vx[3] := 60; vy[3] := 25; vx[4] := 80; vy[4] := 40; vx[5] := 35; vy[5] := 50; vx[6] := 10; vy[6] := 40; gfx_Polygon(7, vx, vy, RED); repeat forever endfunc</pre> <p>This example draws a simple polygon</p>	

2.6.13. `gfx_Triangle(x1, y1, x2, y2, x3, y3, colour)`

Syntax	<code>gfx_Triangle(x1, y1, x2, y2, x3, y3, colour);</code>	
Arguments	<code>x1, y1, x2, y2, x3, y3, colour</code>	
	<code>x1, y1</code>	Specifies the first vertices of the triangle.
	<code>x2, y2</code>	Specifies the second vertices of the triangle.
	<code>x3, y3</code>	Specifies the third vertices of the triangle.
	<code>colour</code>	Specifies the colour for the triangle.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	Draws a triangle outline between vertices <code>x1,y1</code> , <code>x2,y2</code> and <code>x3,y3</code> using the specified colour. The line may be tessellated with the <code>gfx_LinePattern(...)</code> function. Vertices must be specified in an anti-clockwise fashion.	
Example	<code>gfx_Triangle(10,10,30,10,20,30,CYAN);</code>	
	This example draws a CYAN triangular outline with vertices at 10,10 30,10 20,30	

2.6.14. gfx_Dot()

Syntax	<code>gfx_Dot();</code>
Arguments	none
Returns	nothing
Description	Draws a pixel at the current origin using the current object colour.
Example	<pre>gfx_MoveTo(40,50); gfx_ObjectColour(0xRED); gfx_Dot();</pre> <p>This example draws a RED pixel at 40,50</p>

2.6.15. gfx_Bullet(radius)

Syntax	<code>gfx_Bullet(radius);</code>	
Arguments	radius	
	rad	Specifies the radius of the bullet.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Draws a circle or 'bullet point' with radius <i>r</i> at at the current origin using the current object colour.</p> <p>Note: The default PEN_SIZE is set to OUTLINE, however, if PEN_SIZE is set to SOLID, the circle will be drawn filled, if PEN_SIZE is set to OUTLINE, the circle will be drawn as an outline. If the circle is drawn as SOLID, the outline colour can be specified with gfx_OutlineColour(...).</p>	
Example	<pre>gfx_MoveTo(30, 30); gfx_Bullet(10); // Draw a 10pixel radius Bullet at x=30, y=30.</pre>	

2.6.16. `gfx_OrbitInit(&x_dest, &y_dest)`

Syntax	<code>gfx_OrbitInit(&x_dest, &y_dest);</code>	
Arguments	<code>x_dest, y_dest</code>	
	<code>x_dest</code>	Specifies the addresses of the storage locations for the calculated Orbit X-coordinate.
	<code>y_dest</code>	Specifies the addresses of the storage locations for the calculated Orbit Y-coordinate.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Sets up the internal pointers for the <code>gfx_Orbit(..)</code> result variables. The <code>&x_orb</code> and <code>&y_orb</code> parameters are the addresses of the variables or array elements that are used to store the result from the <code>gfx_Orbit(..)</code> function.	
Example	<pre>var targetX, targetY; gfx_OrbitInit(&targetX, &targetY);</pre> <p>This example sets the variables that will receive the result from a <code>gfx_Orbit(..)</code> function call</p>	

2.6.17. `gfx_Orbit(angle, distance)`

Syntax	<code>gfx_Orbit(angle, distance);</code>	
Arguments	angle, distance	
	angle	Specifies the angle from the origin to the remote point. The angle is specified in degrees.
	distance	Specifies the distance from the origin to the remote point in pixel units.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
	Note: result is stored in the variables that were specified with the <code>gfx_OrbitInit(..)</code> function.	
Description	Sets Prior to using this function, the destination address of variables for the calculated coordinates must be set using the <code>gfx_OrbitInit(..)</code> function. The <code>gfx_Orbit(..)</code> function calculates the x, y coordinates of a distant point relative to the current origin, where the only known parameters are the <i>angle</i> and the <i>distance</i> from the current origin. The new coordinates are calculated and then placed in the destination variables that have been previously set with the <code>gfx_OrbitInit(..)</code> function.	
Example	<pre>var targetX, targetY; gfx_OrbitInit(&targetX, &targetY); gfx_MoveTo(30, 30); gfx_Bullet(5) // mark the start point with a small WHITE circle gfx_Orbit(30, 50); // calculate a point 50 pixels away from origin at // 30 degrees gfx_CircleFilled(targetX,targetY,3,0xF800); // mark the target point // with a RED circle</pre> <p>See example comments for explanation.</p>	

2.6.18. gfx_PutPixel(x, y, colour)

Syntax	<code>gfx_PutPixel(x, y, colour);</code>	
Arguments	<code>x, y, colour</code>	
	<code>x, y</code>	Specifies the screen coordinates of the pixel.
	<code>colour</code>	Specifies the colour of the pixel.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Draws a pixel at position x,y using the specified colour.	
Example	<code>gfx_PutPixel(32, 32, 0xFFFF);</code>	
	This example draws a WHITE pixel at x=32, y=32	

2.6.19. gfx_GetPixel(x, y)

Syntax	<code>gfx_GetPixel(x, y);</code>	
Arguments	<code>x, y</code>	
	<code>x, y</code>	Specifies the screen coordinates of the pixel colour to be returned.
	The arguments can be a variable, array element, expression or constant	
Returns	colour	
	<code>colour</code>	The 8 or 16bit colour of the pixel (default 16bit).
Description	Reads the colour value of the pixel at position x,y.	
Example	<pre>gfx_PutPixel(20, 20, 1234); r := gfx_GetPixel(20, 20); print(r);</pre> <p>This example print 1234, the colour of the pixel that was previously placed.</p>	

2.6.20. gfx_MoveTo(xpos, ypos)

Syntax	<code>gfx_MoveTo(xpos, ypos);</code>	
Arguments	xpos, ypos	
	xpos	Specifies the horizontal position of the new origin.
	ypos	Specifies the vertical position of the new origin.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Moves the origin to a new position.	
Example	<pre>#inherit "4DGL_16bitColours.fnc" func help() var x, y, state; print("TOUCHE ME"); touch_Set(TOUCH_ENABLE); // lets enable the touch screen while(touch_Get(TOUCH_STATUS) != TOUCH_PRESSED); //Wait for touch // we'll need a place on the screen to start with gfx_MoveTo(touch_Get(TOUCH_GETX), touch_Get(TOUCH_GETY)); gfx_Set(OBJECT_COLOUR, WHITE); // this will be our line colour while(1) state := touch_Get(TOUCH_STATUS); // Look for touch activity x := touch_Get(TOUCH_GETX); // Grab x and the y := touch_Get(TOUCH_GETY); // y coordinates of the touch if(state == TOUCH_PRESSED) // if there's a press gfx_LineTo(x, y); // Draw a line from previous spot endif if(state == TOUCH_RELEASED) // if there's a release; gfx_CircleFilled(x, y, 10, RED); // Draw a solid red circle endif if(state == TOUCH_MOVING) // if there's movement gfx_PutPixel(x, y, LIGHTGREEN); // we'll draw a green pixel endif wend // Repeat forever endfunc</pre>	

2.6.21. gfx_MoveRel(xoffset, yoffset)

Syntax	<code>gfx_MoveRel(xoffset, yoffset);</code>	
Arguments	<code>xoffset, yoffset</code>	
	xoffset	Specifies the horizontal offset of the new origin.
	yoffset	Specifies the vertical offset of the new origin.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Moves the origin to a new position relative to the old position.	
Example	<pre>gfx_MoveTo(10, 20); gfx_MoveRel(-5, -3); gfx_Dot();</pre> <p>This example draws a pixel using the current object colour at x=5, y=17</p>	

2.6.22. gfx_IncX()

Syntax	<code>gfx_IncX();</code>	
Arguments	none	
Returns	<code>old_origin</code>	
	<code>old_origin</code>	Returns the current X origin before the increment.
Description	Increment the current X origin by 1 pixel unit. The original value is returned before incrementing. The return value can be useful if a function requires the current point before insetting occurs.	
Example	<pre>var n; gfx_MoveTo(20,20); n := 96; while (n--)</pre> <div style="background-color: #e6f2e6; padding: 2px;"> <pre> gfx_ObjectColour(n/3);</pre> </div> <div style="background-color: #e6f2e6; padding: 2px;"> <pre> gfx_Bullet(2);</pre> </div> <div style="background-color: #e6f2e6; padding: 2px;"> <pre> gfx_IncX();</pre> </div> <pre>wend</pre> <p>This example draws a simple rounded vertical gradient.</p>	

2.6.23. gfx_IncY()

Syntax	<code>gfx_IncY();</code>	
Arguments	none	
Returns	<code>old_Yorigin</code>	Returns the current Y origin before the increment.
Description	Increment the current Y origin by 1 pixel unit. The original value is returned before incrementing. The return value can be useful if a function requires the current point before insetting occurs.	
Example	<pre>var n; gfx_MoveTo(20,20); n := 96; while (n--) gfx_ObjectColour(n/3); gfx_LineRel(20, 0); gfx_IncY(); wend</pre> <p>This example draws a simple horizontal gradient using lines.</p>	

2.6.24. gfx_LineTo(xpos, ypos)

Syntax	<code>gfx_LineTo(xpos, ypos);</code>	
Arguments	xpos, ypos	
	xpos	Specifies the horizontal position of the line end as well as the new origin.
	ypos	Specifies the vertical position of the line end as well as the new origin.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Draws a line from the current origin to a new position. The Origin is then set to the new position. The line is drawn using the current object colour. The line may be tessellated with the <code>gfx_LinePattern(...)</code> function.	
Example	<pre>gfx_MoveTo(10, 20); gfx_LineTo(60, 70);</pre> <p>This example draws a line using the current object colour between x1=10,y1=20 and x2=60,y2=70. The new origin is now set at x=60,y=70.</p>	

2.6.25. `gfx_LineRel(xpos, ypos)`

Syntax	<code>gfx_LineRel(xpos, ypos);</code>	
Arguments	<code>xpos, ypos</code>	
	<code>xpos</code>	Specifies the horizontal end point of the line.
	<code>ypos</code>	Specifies the vertical end point of the line.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Draws a line from the current origin to a new position. The line is drawn using the current object colour. The current origin is not altered. The line may be tessellated with the <code>gfx_LinePattern(...)</code> function.	
Example	<pre>gfx_LinePattern(0b1100110011001100); gfx_MoveTo(10, 20); gfx_LineRel(50, 50);</pre> <p>This example draws a tessellated line using the current object colour between 10,20 and 50,50. Note: that <code>gfx_LinePattern(0)</code>; must be used after this to return line drawing to normal solid lines.</p>	

2.6.26. gfx_BoxTo(x2, y2)

Syntax	<code>gfx_BoxTo(x2, y2);</code>	
Arguments	<code>x2, y2</code>	
	<code>x2,y2</code>	Specifies the diagonally opposed corner of the rectangle to be drawn, the top left corner (assumed to be x1, y1) is anchored by the current origin.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Draws a rectangle from the current origin to the new point using the current object colour. The top left corner is anchored by the current origin (x1, y1), the bottom right corner is specified by x2, y2.</p> <p>Note: The default PEN_SIZE is set to OUTLINE, however, if PEN_SIZE is set to SOLID, the rectangle will be drawn filled, if PEN_SIZE is set to OUTLINE, the rectangle will be drawn as an outline. If the circle is drawn as SOLID, the outline colour can be specified with <code>gfx_OutlineColour(...)</code>. If OUTLINE_COLOUR is set to 0, no outline is drawn.</p>	
Example	<pre>gfx_MoveTo(40,40); n := 10; while (n--) gfx_BoxTo(50,50); gfx_BoxTo(30,30); wend</pre> <p>This example draws 2 boxes, anchored from the current origin.</p>	

2.6.27. gfx_SetClipRegion()

Syntax	<code>gfx_SetClipRegion();</code>
Arguments	none
Returns	nothing
Description	Forces the clip region to the extent of the last text that was printed, or the last image that was shown.
Example	

2.6.28. gfx_Ellipse(x, y, xrad, yrad, colour)

Syntax	<code>gfx_Ellipse(x, y, xrad, yrad, colour);</code>	
Arguments	<code>x, y, xrad, yrad, colour</code>	
	x, y	specifies the horizontal and vertical position of the centre of ellipse
	xrad, yrad	Specifies x-radius and y-radius of the ellipse.
	colour	Specifies the colour for the lines
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Plots a coloured Ellipse on the screen at centre x,y with xradius = xrad and yradius = yrad. if PenSize = 0 Ellipse is Solid if PenSize = 1 Ellipse is Outline	
Example	<code>gfx_Ellipse(200, 80, 5, 10, YELLOW);</code>	

2.6.29. gfx_EllipseFilled(x, y, xrad, yrad, colour)

Syntax	<code>gfx_EllipseFilled(x, y, xrad, yrad, colour);</code>	
Arguments	<code>x, y, xrad, yrad, colour</code>	
	x, y	specifies the horizontal and vertical position of the centre of ellipse
	xrad, yrad	Specifies x-radius and y-radius of the ellipse.
	colour	Specifies the colour for the lines
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Plots a solid coloured Ellipse on the screen at centre x,y with xradius = xrad and yradius = yrad.	
Example	<code>gfx_EllipseFilled(200,110,10,5, GREEN) ;</code>	

2.6.30. gfx_Button(state, x, y, buttonColour, txtColour, font, txtWidth, txtHeight, text)

Syntax	<code>gfx_Button(state, x, y, buttonColour, txtColour, font, txtWidth, txtHeight, text);</code>	
Arguments	<code>state, x, y, buttonColour, txtColour, font, txtWidth, txtHeight, text</code>	
	state	0 = Button pressed; 1 = Button raised.
	x, y	Specifies the top left corner position of the button on the screen.
	buttonColour	Button colour
	txtColour	Text Colour
	font	Specifies the Font ID.
	txtWidth	Specifies the width of the text. This value is the font width multiplier and minimum value must be 1.
	txtHeight	Specifies the height of the text. This value is the font height multiplier and minimum value must be 1.
	text	Specifies the text string. The text string must be within the range of printable ascii character set. The string may have \n characters embedded to create a multiline button.
Returns	nothing	
Description	<p>Draws a 3 dimensional Text Button at screen location defined by x, y parameters (top left corner). The size of the button depends on the font, width, height and length of the text. The button can contain multiple lines of text by having the \n character embedded in the string for the end of line marker. In this case, the widest text in the string sets the overall width, and the height of the button is set by the number of text lines. In the case of multiple lines, each line is left justified. If you wish to centre or right justify the text, you will need to prepare the text string according to your requirements.</p>	
Example	<pre>#constant LEFT 30 #constant TOP 150 #constant TEXTWIDTH 2 #constant TEXTHEIGHT 2 //----- func main() // Draw a button as a Text Box (indented) gfx_Button(DOWN, 0, 30, GREEN, WHITE, FONT_4, TEXTWIDTH, TEXTHEIGHT, "4DGL-Demo"); touch_Set(TOUCH_ENABLE); repeat // Draw the Push Button (raised) gfx_Button(UP, LEFT, TOP, BLUE, RED, FONT_4, TEXTWIDTH, TEXTHEIGHT, " PRESS "); // set touch detect region to that of the push button touch_DetectRegion(LEFT, TOP, gfx_Get(RIGHT_POS), gfx_Get(BOTTOM_POS)); // Wait until the button is pressed</pre>	

```
while(touch_Get(TOUCH_STATUS) != TOUCH_PRESS);  
  
// now redraw the Push Button (depressed)  
gfx_Button(DOWN, LEFT, TOP, BLUE, WHITE, FONT_4, TEXTWIDTH,  
TEXTHEIGHT, " PRESS ");  
  
// Wait until the button is pressed  
while(touch_Get(TOUCH_STATUS) != TOUCH_RELEASE);  
forever  
endfunc
```

2.6.31. `gfx_Button2(state, x, y, width, height, buttonColour, txtColour, text)`

Syntax	<code>gfx_Button(mode, x, y, width, height, buttoncolour, textcolour, text);</code>	
Arguments	mode, x, y, width, height, buttoncolour, textcolour, text	
	mode	0 = Button pressed; 1 = Button raised.
	x, y	Specifies the top left corner position of the button on the screen.
	width	Specifies the width of the button.
	height	Specifies the height of the button.
	buttonColour	Button colour
	txtColour	Text Colour
	text	Specifies the text string. The text string must be within the range of printable ascii character set. The string may have \n characters embedded to create a multiline button.
Returns	nothing	
Description	Draws a 3 dimensional Text Button at screen location defined by x, y parameters (top left corner). The size of the button is defined by the width and height parameters. The text is centred within those bounds. The button has square corners.	
Example	<pre> #constant LEFT 30 #constant TOP 150 #constant BWIDTH 50 #constant BHEIGHT 50 //----- func main() touch_Set(TOUCH_ENABLE); repeat // Draw the Push Button (raised) gfx_Button(UP, LEFT, TOP, BWIDTH, BHEIGHT, BLUE, RED, " PRESS "); // set touch detect region to that of the push button touch_DetectRegion(LEFT, TOP, gfx_Get(RIGHT_POS), gfx_Get(BOTTOM_POS)); // Wait until the button is pressed while(touch_Get(TOUCH_STATUS) != TOUCH_PRESS); // now redraw the Push Button (depressed) gfx_Button(DOWN, LEFT, TOP, BWIDTH, BHEIGHT, BLUE, RED, " PRESS "); // Wait until the button is pressed while(touch_Get(TOUCH_STATUS) != TOUCH_RELEASE); forever endfunc </pre>	

2.6.32. `gfx_Button3(state, x, y, width, height, buttonColour, txtColour, text)`

Syntax	<code>gfx_Button(mode, x, y, width, height, buttoncolour, textcolour, text);</code>	
Arguments	mode, x, y, width, height, buttoncolour, textcolour, text	
	mode	0 = Button pressed; 1 = Button raised.
	x, y	Specifies the top left corner position of the button on the screen.
	width	Specifies the width of the button.
	height	Specifies the height of the button.
	buttonColour	Button colour
	txtColour	Text Colour
	text	Specifies the text string. The text string must be within the range of printable ascii character set. The string may have \n characters embedded to create a multiline button.
Returns	nothing	
Description	Draws a 3 dimensional Text Button at screen location defined by x, y parameters (top left corner). The size of the button is defined by the width and height parameters. The text is centred within those bounds. The button has rounded corners depending <code>gfx_BevelRadius()</code> .	
Example	<pre> #constant LEFT 30 #constant TOP 150 #constant BWIDTH 50 #constant BHEIGHT 50 //----- func main() touch_Set(TOUCH_ENABLE); repeat // Draw the Push Button (raised) gfx_Button(UP, LEFT, TOP, BWIDTH, BHEIGHT, BLUE, RED, " PRESS "); // set touch detect region to that of the push button touch_DetectRegion(LEFT, TOP, gfx_Get(RIGHT_POS), gfx_Get(BOTTOM_POS)); // Wait until the button is pressed while(touch_Get(TOUCH_STATUS) != TOUCH_PRESS); // now redraw the Push Button (depressed) gfx_Button(DOWN, LEFT, TOP, BWIDTH, BHEIGHT, BLUE, RED, " PRESS "); // Wait until the button is pressed while(touch_Get(TOUCH_STATUS) != TOUCH_RELEASE); forever endfunc </pre>	

2.6.33. gfx_Panel(state, x, y, width, height, Colour)

Syntax	<code>gfx_Panel(state, x, y, width, height, Colour);</code>	
Arguments	state, x, y, width, height, colour	
	state	0 = recessed; 1 = raised.
	x, y	Specifies the top left corner position of the panel on the screen.
	width	Specifies the width of the panel.
	height	Specifies the Height of the panel.
	Colour	Specifies the colour of the panel.
Returns	nothing	
Description	Draws a 3 dimensional rectangular panel at a screen location defined by x, y parameters (top left corner). The size of the panel is set with the width and height parameters. The colour is defined by colour The state parameter determines the appearance of the panel, 0 = recessed, 1 = raised.	
Example	<pre>#constant LEFT 15 #constant TOP 15 #constant WIDTH 100 #constant HEIGHT 100 func main() // Draw a panel gfx_Panel(RAISED, LEFT, TOP, WIDTH, HEIGHT, GRAY); repeat forever endfunc</pre>	

2.6.34. gfx_RoundPanel(state, x, y, width, height, radius, bevelwidth, Colour)

Syntax	<code>gfx_Panel(state, x, y, width, height, radius, bevelwidth, Colour);</code>	
Arguments	state, x, y, width, height, radius, bevelwidth, Colour	
	state	0 = recessed; 1 = raised; 2 = hide (draw object in background colour)
	x, y	Specifies the top left corner position of the panel on the screen.
	width	Specifies the width of the panel.
	height	Specifies the Height of the panel.
	radius	Specifies the corner radius.
	bevelwidth	Set Panel bevel width 0-15 pixels.
	Colour	Specifies the colour of the panel.
Returns	nothing	
Description	<p>Draws a 3 dimensional rounded rectangular panel at a screen location defined by x, y parameters (top left corner). Width and height may be zero allowing the function to be used for rounded panels, rounded buttons, and circular buttons.</p> <p>Bounding rectangle is x1-radius-bevelwidth, y1-radius-bevelwidth, x2+radius+bevelwidth, y2+radius+bevelwidth.</p>	
Example	<code>gfx_RoundPanel(PANEL_RAISED, 100, 100, 30, 20, GRAY);</code>	

2.6.35. `gfx_Slider(mode, x1, y1, x2, y2, colour, scale, value)`

Syntax	<code>gfx_Slider(mode, x1, y1, x2, y2, colour, scale, value);</code>	
Arguments	mode, x1, y1, x2, y2, colour, scale, value	
	mode	mode = 0 : Slider Indented, mode = 1 : Slider Raised, mode 2, Slider Hidden (background colour).
	x1, y1	Specifies the top left corner position of the slider on the screen.
	x2, y2	Specifies the bottom right corner position of the slider on the screen.
	colour	Specifies the colour of the Slider bar.
	Scale	scale = n : sets the full scale range of the slider for the thumb from 0 to n.
	Value	if value positive, sets the relative position of the thumb on the slider bar, else set thumb to ABS position of the negative number.
Returns	<p>If the value parameter was a positive number (i.e:- value is a proportion of the scale parameter), the true (implied x or y axis) position of the thumb is returned.</p> <p>If the value parameter was a negative number (i.e:- thumb is being set to an ABSolute graphics position), the actual slider value (which is a proportion of the scale parameter) is returned.</p>	
Description	<p>Draws a vertical or horizontal slider bar on the screen. The <code>gfx_Slider</code> function has several different modes of operation. In order to minimise the amount of graphics functions we need, all modes of operation are selected naturally depending on the parameter values.</p> <p>Selection rules:</p> <p>1a] if $x_2 - x_1 > y_2 - y_1$ slider is assumed to be horizontal (ie: if width > height, slider is horizontal)</p> <p>1b] if $x_2 - x_1 \leq y_2 - y_1$ slider is assumed to be vertical (ie: if height \leq width, slider is horizontal)</p> <p>2a] If value is positive, thumb is set to the position that is the proportion of value to the scale parameter.(used to set the control to the actual value of a variable)</p> <p>2b] If value is negative, thumb is driven to the graphics position set by the ABSolute of value value. (used to set thumb to its actual graphical position (usually by touch screen)</p> <p>3] The thumb colour is determine by <code>gfx_Set(OBJECT_COLOUR, value);</code> , however, if the current object colour is BLACK, a darkened shade of the colour parameter is used for the thumb .</p>	
	<pre>func drawRedSlider() gfx_Slider(0, rSlider[0], rSlider[1], rSlider[2], rSlider[3], RED, 255, valR); txt_MoveCursor(1, 12); txt_Set(TEXT_OPACITY, OPAQUE); txt_Set(TEXT_COLOUR, RED); print (" "); txt_MoveCursor(1, 12); print ([DEC] valR); endfunc</pre>	

2.6.36. `gfx_Slider2(mode, x1, y1, width, height, colour, scale, value)`

Syntax	<code>gfx_Slider(mode, x1, y1, width, height, colour, scale, value);</code>	
Arguments	mode, x1, y1, x2, y2, colour, scale, value	
	mode	mode = 0 : Slider Indented, mode = 1 : Slider Raised, mode 2, Slider Hidden (background colour).
	x1, y1	Specifies the top left corner position of the slider on the screen.
	width	Specifies the width of the slider on the screen.
	height	Specifies the height of the slider on the screen.
	colour	Specifies the colour of the Slider bar.
	scale	scale = n : sets the full scale range of the slider for the thumb from 0 to n.
	value	value = m : sets the relative position of the thumb $0 \leq m \leq n$
Returns	<p>If the value parameter was a positive number (i.e:- value is a proportion of the scale parameter), the true (implied x or y axis) position of the thumb is returned.</p> <p>If the value parameter was a negative number (i.e:- thumb is being set to an ABSolute graphics position), the actual slider value (which is a proportion of the scale parameter) is returned.</p>	
Description	<p>Draws a vertical or horizontal slider bar on the screen. The <code>gfx_Slider</code> function has several different modes of operation. In order to minimise the amount of graphics functions we need, all modes of operation are selected naturally depending on the parameter values.</p> <p>Selection rules:</p> <p>1a] if width > height, slider is horizontal</p> <p>1b] if height <= width, slider is horizontal</p> <p>2a] If value is positive, thumb is set to the position that is the proportion of value to the scale parameter.(used to set the control to the actual value of a variable)</p> <p>2b] If value is negative, thumb is driven to the graphics position set by the ABSolute of value value. (used to set thumb to its actual graphical position (usually by touch screen)</p> <p>3] The thumb colour is determine by <code>gfx_Set(OBJECT_COLOUR, value);</code> , however, if the current object colour is BLACK, a darkened shade of the colour parameter is used for the thumb .</p>	
	<pre>func drawRedSlider () gfx_Slider(0, rSlider[0], rSlider[1], rSlider[2], rSlider[3], RED, 255, valR); txt_MoveCursor(1, 12); txt_Set(TEXT_OPACITY, OPAQUE); txt_Set(TEXT_COLOUR, RED); print (" "); txt_MoveCursor(1, 12); print ([DEC] valR); endfunc</pre>	

2.6.37. gfx_ScreenCopyPaste(xs, ys, xd, yd, width, height)

Syntax	gfx_ScreenCopyPaste(xs, ys, xd, yd, width, height);	
Arguments	xs, ys, xd, yd, width, height	
	xs, ys	Specifies the horizontal and vertical position of the top left corner of the area to be copied (source).
	xd, yd	Specifies the horizontal and vertical position of the top left corner of where the paste is to be made (destination).
	width	Specifies the width of the copied area.
	height	Specifies the height of the copied area.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Copies an area of a screen from xs, ys of size given by width and height parameters and pastes it to another location determined by xd, yd.	

2.6.38. gfx_RGBto565(RED, GREEN, BLUE)

Syntax	<code>gfx_RGBto565(RED, GREEN, BLUE);</code>	
Arguments	RED, GREEN, BLUE	
	RED	8bit colour value for RED.
	GREEN	8bit colour value for GREEN. .
	BLUE	8bit colour value for BLUE.
	The arguments can be a variable, array element, expression or constant	
Returns	Returns the 16bit (RED: 5, GREEN: 6, BLUE: 5 format) colour value.	
Description	Returns the 16bit (RED: 5, GREEN: 6, BLUE: 5 format) colour value of a 24bit (RED: 8, GREEN: 8, BLUE: 8 format) colour.	

2.6.39. gfx_332to565(COLOUR8BIT)

Syntax	<code>gfx_332to565(COLOUR8BIT);</code>	
Arguments	COLOUR8BIT	
	COLOUR8BIT	8bit colour value. 3bits for RED, 3bits for GREEN, 2bits for BLUE.
Returns	Returns the 16bit (RED: 5, GREEN: 6, BLUE: 5 format) value	
Description	Returns the 16bit (RED: 5, GREEN: 6, BLUE: 5 format) value of an 8bit (RED: 3, GREEN: 3, BLUE: 2 format) colour	

2.6.40. gfx_565to332(COLOUR)

Syntax	gfx_565to332(COLOUR);	
Arguments	COLOUR16BIT	
	COLOUR16BIT	16bit colour value. 5bits for RED, 6bits for GREEN, 5bits for BLUE.
Returns	Returns the 8bit (RED: 3, GREEN: 3, BLUE: 2 format) value	
Description	Returns the 8bit (RED: 3, GREEN: 3, BLUE: 2 format) value of a 16bit (RED: 5, GREEN: 6, BLUE: 5 format) colour.	

2.6.41. gfx_TriangleFilled(x1, y1, x2, y2, x3, y3, colour)

Syntax	<code>gfx_TriangleFilled(x1, y1, x2, y2, x3, y3, colour);</code>	
Arguments	<code>x1, y1, x2, y2, x3, y3, colour</code>	
	<code>x1, y1</code>	Specifies the first vertices of the triangle.
	<code>x2, y2</code>	Specifies the second vertices of the triangle.
	<code>x3, y3</code>	Specifies the third vertices of the triangle.
	<code>colour</code>	Specifies the colour for the triangle.
	The arguments can be a variable, array element, expression or constant	
Returns	<code>nothing</code>	
Description	Draws a Solid triangle between vertices <code>x1,y1</code> , <code>x2,y2</code> and <code>x3,y3</code> using the specified colour. Vertices must be specified in an anti-clockwise fashion.	
Example	<pre>gfx_TriangleFilled(10,10,30,10,20,30,CYAN);</pre> <p>This example draws a CYAN Solid triangle with vertices at 10,10 30,10 20,30</p>	

2.6.42. gfx_PolygonFilled(n, vx, vy, colour)

Syntax	<code>gfx_PolygonFilled(n, vx, vy, colour);</code>	
Arguments	n, vx, vy, colour	
	n	Specifies the number of elements in the x and y arrays specifying the vertices for the polygon.
	vx	Specifies the addresses of the storage of the array of elements for the x coordinates of the vertices.
	vy	Specifies the addresses of the storage of the array of elements for the y coordinates of the vertices.
	colour	Specifies the colour for the polygon
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Draws a solid Polygon between specified vertices: x1,y1 x2,y2 ... xn,yn using the specified colour. The last point is drawn back to the first point, completing the polygon. Vertices must be minimum of 3 and can be specified in any fashion.	
Example	<pre> var vx[7], vy[7]; func main() vx[0] := 10; vy[0] := 10; vx[1] := 35; vy[1] := 5; vx[2] := 80; vy[2] := 10; vx[3] := 60; vy[3] := 25; vx[4] := 80; vy[4] := 40; vx[5] := 35; vy[5] := 50; vx[6] := 10; vy[6] := 40; gfx_PolygonFilled(7, vx, vy, RED); repeat forever endfunc </pre> <p>This example draws a simple filled polygon</p>	

2.6.43. gfx-Origin(x, y)

Syntax	<code>gfx-Origin(x, y);</code>	
Arguments	<code>x, y</code>	
	<code>x, y</code>	Specifies the horizontal and vertical position of the top left corner of the clipping window.
Returns	nothing	
Description	Sets relative screen offset for horizontal and vertical for the top left corner for graphics objects.	
Example	<code>gfx_Offset(arg1, arg2);</code>	

2.6.44. gfx_Get(mode)

Syntax	<code>gfx_Get(mode);</code>	
Arguments	mode	
	mode	<p>mode = 0 : Current orientations Max X Value (X_MAX) mode = 1 : Current orientations Max Y Value (Y_MAX) mode = 2 : Left location of Object mode = 3 : Top location of Object mode = 4 : Right location of Object mode = 5 : Bottom location of Object mode = 6 : Get current internal X position mode = 7 : Get current internal Y position</p>
Returns	<p>Mode0 Returns the maximum horizontal value of the display.</p> <p>Mode1 Returns the maximum vertical value of the display.</p> <p>Mode2 Returns the left location of the last drawn object such as a slider or button or an image/video.</p> <p>Mode3 Returns the top location of the last drawn object such as a slider or button or an image/video.</p> <p>Mode4 Returns the right location of the last drawn object such as a slider or button or an image/video.</p> <p>Mode5 Returns the bottom location of the last drawn object such as a slider or button or an image/video.</p> <p>Mode6 Returns the internal X position that was set with MoveTo(x, y); or gfx_Set(X_ORG, pos);</p> <p>Mode7 Returns the internal Y position that was set with MoveTo(x, y); or gfx_Set(X_ORG, pos);</p>	
Description	Returns various graphics parameters to caller.	
Example	<pre>var := gfx_Get(X_MAX); //Returns the maximum horizontal resolution of the display var := gfx_Get(0); var := gfx_Get(Y_MAX); //Returns the maximum vertical resolution of the display var := gfx_Get(1); var := gfx_Get(RIGHT_POS); //Returns the right location of the last drawn object //that only has top, left parameters such as a button // or an image/video. var := gfx_Get(2); var := gfx_Get(BOTTOM_POS); //Returns the bottom location of the last drawn object //that only has top, left parameters such as a button //or an image/video. var := gfx_Get(3);</pre>	

2.6.45. `gfx_ClipWindow(x1, y1, x2, y2)`

Syntax	<code>gfx_ClipWindow(x1, y1, x2, y2);</code>	
Arguments	<code>x1, y1, x2, y2</code>	
	<code>x1, y1</code>	Specifies the horizontal and vertical position of the top left corner of the clipping window.
	<code>x2, y2</code>	Specifies the horizontal and vertical position of the bottom right corner of the clipping window.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Specifies a clipping window region on the screen such that any objects and text placed onto the screen will be clipped and displayed only within that region. For the clipping window to take effect, "Clipping" setting must be enabled separately using <code>gfx_Set(CLIPPING, ON)</code> or the shortcut <code>gfx_Clippping(ON)</code> .	
Example	<pre>var n; gfx_ClipWindow(10, 10, 50, 50) n := 50000; while (n--) gfx_PutPixel (RAND() %100, RAND() %100, RAND()); wend repeat forever</pre> <p>This example will draw 50000 random colour pixels, only the pixels within the clipping area will be visible</p>	

2.6.46. **gfx_Set(function, value)**

Syntax	gfx_Set(function, value);	
Arguments	function, value	
	function	The function number determines the required action for various graphics control functions. Usually a constant, but can be a variable, array element, or expression. There are pre-defined constants for each of the functions.
	value	A variable, array element, expression or constant holding a value for the selected function.
Returns	nothing	
Description	<p>Given a function number and a value, set the required graphics control parameter, such as size, colour, and other parameters. (See the Single parameter short-cuts for the gfx_Set functions below). It is strongly recommended to use the pre-defined constants rather than the mode numbers.</p> <p>Note: Although it is often required to be able to set graphics functions with a single function call for graphics engine related functions, there is a complete set of single parameter shortcut functions that have exactly the same function as each of the gfx_Set modes and saves 1 parameter, i.e. uses less memory.</p>	
	function	value
Predefined Name	Description	
PEN_SIZE	Set the draw mode for gfx_LineTo, gfx_LineRel, gfx_Dot, gfx_Bullet and gfx_BoxTo (default mode is OUTLINE) nb:- pen size is set to OUTLINE for normal operation	0 or SOLID 1 or OUTLINE
BACKGROUND_COLOUR	Set the screen background colour	Colour, 0-65535
OBJECT_COLOUR	Generic colour for gfx_LineTo(...), gfx_LineRel(...), gfx_Dot(), gfx_Bullet(...) and gfx_BoxTo(...)	Colour, 0-65535
CLIPPING	Turns clipping on/off. The clipping points are set with gfx_ClipWindow(...) and must be set first.	1 or 0 (ON or OFF)
TRANSPARENT_COLOUR	Colour that needs to be made transparent.	Colour, 0-65535
TRANSPARENCY	Turn the transparency ON or OFF. Transparency is automatically turned OFF after the next image or video command.	1 or 0 (ON or OFF)
FRAME_DELAY	Set the inter frame delay for media_Video(...)	0 to 255msec
SCREEN_MODE	Set required screen behaviour/orientation.	0 or LANDSCAPE 1 or LANDSCAPE_R 2 or PORTRAIT 3 or PORTRAIT_R
OUTLINE_COLOUR	Outline colour for rectangles and circles (set to 0 for no effect)	Colour, 0-65535
CONTRAST	LCD MODULES: contrast 0 = display OFF, 1-15 = contrast level (Actually backlight brightness)	0 or OFF 1 to 15 for levels
LINE_PATTERN	Sets the line draw pattern for line drawing. If set to zero,	0 or OFF

	lines are solid, else each '1' bit represents a pixel that is turned off. Example: gfx_Set(LINE_PATTERN, 0b1111000011110000); // draw dotted line	1 to 0xFFFF 0 bits for pixels on 1 bits for pixels off
COLOUR_MODE	Sets 8 or 16bit colour mode Function not available, fixed as 16bit mode.	0 or COLOUR16 1 or COLOUR8
BEVEL_WIDTH	Set Button Bevel Width, 0 pixel to 15pixels.	0 None 1 to 15 pixels
BEVEL_SHADOW	graphics button bevel shadow depth	0 None 1 to 15 pixels
X_ORIGIN	sets the origin of drawn objects to a position other than 0,0	
Y_ORIGIN	sets the origin of drawn objects to a position other than 0,0	
DISPLAY_PAGE	Choose Page to be displayed. Applies to uLCD-43D* only.	0, 1 or 2.
READ_PAGE	Choose Page to be read. Applies to uLCD-43D* only.	0, 1 or 2.
WRITE_PAGE	Choose Page to be written. Applies to uLCD-43D* only.	0, 1 or 2.

Single parameter short-cuts for the gfx_Set(..) functions

Function Syntax	Function Action	value
gfx_PenSize(mode)	Set the draw mode for gfx_LineTo, gfx_LineRel, gfx_Dot, gfx_Bullet and gfx_BoxTo Note: pen size is set to OUTLINE for normal operation (default).	0 or SOLID 1 or OUTLINE
gfx_BGcolour(colour)	Set the screen background colour	Colour 0-65535
gfx_ObjectColour(colour)	Generic colour for gfx_LineTo(...), gfx_LineRel(...), gfx_Dot(), gfx_Bullet(...) and gfx_BoxTo	Colour 0-65535
gfx_Clippping(mode)	Turns clipping on/off. The clipping points are set with gfx_ClipWindow(...)	0 or 1 (ON or OFF)
gfx_TransparentColour(colour)	Colour that needs to be made transparent.	Colour, 0-65535
gfx_Transparency(mode)	Turn the transparency ON or OFF.	1 or 0 (ON or OFF)
gfx_FrameDelay(delay)	Set the inter frame delay for media_Video(...)	0 to 255msec
gfx_ScreenMode(mode)	Graphics orientation LANDSCAPE, LANDSCAPE_R, PORTRAIT, PORTRAIT_R	1 or LANDSCAPE 2 or LANDSCAPE_R 3 or PORTRAIT 4 or PORTRAIT_R
gfx_OutlineColour(colour)	Outline colour for rectangles and circles. (set to 0 for no effect)	Colour 0-65535
gfx_Contrast(value)	LCD MODULES: contrast 0 = display OFF, 1-15 = contrast level (Actually backlight brightness)	0 or OFF 1 to 15 for levels
gfx_LinePattern(pattern)	Sets the line draw pattern for line drawing. If set to zero, lines are solid, else each '1' bit represents a pixel that is turned off. See code examples for further reference. Example: gfx_Set(LINE_PATTERN, 0b1111000011110000); // draw dotted line	0 or OFF 1 to 0xFFFF 0 bits for pixels on 1 bits for pixels off
gfx_BevelRadius(radius)	graphics button bevel radius	0 None 1 to 15 pixels
gfx_BevelWidth(mode)	graphics button bevel width	0 None 1 to 15 pixels

gfx_BevelShadow(value)	graphics button bevel shadow depth	0 None 1 to 15 pixels
gfx_Xorigin(offset)	graphics X origin	
gfx_Yorigin(offset)	graphics Y origin	

2.6.47. gfx_Arc(cx, yc, radius, step, startangle, endangle, mode)

Syntax	<code>gfx_Arc(cx, cy, radius, step, startangle, endangle, mode);</code>	
Arguments	<code>cx, cy, radius, step, startangle, endangle, mode</code>	
	cx, cy	Center of the arc.
	radius	Radius of the arc.
	step	Step is the stepping angle increment for the fineness of the arc.
	startangle	Starting angle of the arc.
	endangle	Ending angle of the arc.
	mode	mode = 0, outer circumference line only mode = 1, outer circumference and lines back to cx:cy
Returns	Nothing	
Description	Draws an arc at "xc":"yc" with radius "radius", starting at "startangle" and ending at "endangle". Colour is determined by current object colour.	
Example		

2.6.48. gfx_CheckBox(state, x, y, Width, Height, boxColour, textColour, text)

Syntax	<code>gfx_CheckBox(state, x, y, Width, Height, boxColour, textColour, text);</code>	
Arguments	<code>state, x, y, width, height, boxColour, textColour, text</code>	
	state	state = 1 = UNCHECKED : CheckBox Unchecked state = 0 = CHECKED : Checkbox Checked
	x, y	Top left corner of the Checkbox.
	width	Width of the checkbox.
	height	Height of the checkbox.
	boxColour	Checkbox colour.
	textColour	Text colour.
	text	The text is to the right of the checkbox and truncated if necessary
Returns	Nothing	
Description	Draws a CheckBox at screen location defined by x,y arguments (top left corner).	
Example		

2.6.49. gfx_RadioButton(state, x, y, width, height, boxColour, textColour, text)

Syntax	<code>gfx_RadioButton(state, x, y, width, height, boxColour, textColour, text);</code>	
Arguments	state, x, y, width, height, boxColour, textColour, text	
	state	state = 1 = UNCHECKED : Radio-button Unchecked state = 0 = CHECKED : Radio-button Checked
	x, y	Top left corner of the Radio-button.
	width	Width of the Radio-button.
	height	Height of the Radio-button.
	boxColour	Radio-button colour.
	textColour	Text colour.
	text	The text is to the right of the Radio-button and truncated if necessary
Returns	Nothing	
Description	Draws a Radio-button at screen location defined by x,y arguments (top left corner).	
Example		

2.6.50. gfx_FillPattern(patptr, mode)

Syntax	<code>gfx_FillPattern(patptr, mode);</code>	
Arguments	<code>patptr, mode</code>	
	patptr	0 = Off, 0xFFE0 to 0xFFFF = builtin patterns, else patptr points to a users 8 byte pattern.
	mode	TRANSPARENT or OPAQUE (0 or 1)
Returns	pointer	
	pointer	Returns the handle of the previous pattern
Description	<p>Selects a tessellating pattern for painting solid objects. 'patptr' points to an 8x8 tile for rendering filled areas.</p> <p>Rendering is turned off with <code>gfx_FillPattern(0);</code> or <code>gfx_FillPattern(OFF);</code></p> <p>'mode' maybe TRANSPARENT or OPAQUE (0 or 1), for OPAQUE mode, the current screen colour is used for the 'off' pixels, for transparent mode, the 'off' pixels are not drawn.</p> <p><code>gfx_FillPattern</code> affects all filled object, including polygons. There are 32 builtin patterns; these are obtained using the pre-defined constants <code>FILLPATTERN_0</code> to <code>FILLPATTERN_31</code>. Note that the constants range from 0xFFE0 to 0xFFFF, any other value is assumed to be a pointer to a user's 8 byte block pattern.</p> <p>Predefined constants are used to select the internal patterns, FILLPATTERN_0 through to FILLPATTERN_31</p>	
Example		

2.6.51. `gfx_Gradient(style, x1, y1, x2, y2, color1, color2)`

Syntax	<code>gfx_Gradient(style, x1, y1, x2, y2, color1, color2);</code>	
Arguments	<code>style, x1, y1, x2, y2, color1, color2</code>	
	style	Specifies gradient style. GRAD_DOWN gradient changes in the vertical direction GRAD_RIGHT gradient change in the horizontal direction GRAD_UP gradient changes in the vertical direction GRAD_LEFT gradient change in the horizontal direction GRAD_WAVE_VER gradient wave in the vertical direction GRAD_WAVE_HOR gradient wave in the horizontal direction
	x1, y1	Specifies top left corner of the rectangle.
	x2, y2	Specifies bottom right corner of the rectangle.
	color1	Specifies starting colour.
	color2	Specifies ending colour.
Returns	Nothing	
Description	Draws a graduated colour rectangle at the specified co-ordinate. Rendering can be obtained with <code>gfx_FillPattern(PATTRN)</code> ; or <code>gfx_FillPattern(OFF)</code> ; for no fill pattern.	
Example	<pre>//Draw graduated colour rectangle gfx_Gradient(GRAD_WAVE_HOR, 10, 10, 230, 160, BLACK, WHITE);</pre>	

2.6.52. `gfx_RoundGradient(style, x1, y1, x2, y2, radius, color1, color2)`

Syntax	<code>gfx_RoundGradient(style, x1, y1, x2, y2, radius, color1, color2);</code>	
Arguments	<code>style, x1, y1, x2, y2, radius, color1, color2</code>	
	style	Specifies gradient style. GRAD_DOWN gradient changes in the vertical direction GRAD_RIGHT gradient change in the horizontal direction GRAD_UP gradient changes in the vertical direction GRAD_LEFT gradient change in the horizontal direction GRAD_WAVE_VER gradient wave in the vertical direction GRAD_WAVE_HOR gradient wave in the horizontal direction
	x1, y1	Specifies top left corner of the rectangle.
	x2, y2	Specifies bottom right corner of the rectangle.
	radius	Specifies the corner radius.
	color1	Specifies starting colour.
	color2	Specifies ending colour.
Returns	Nothing	
Description	<p>Draws a graduated colour rounded rectangle at the specified co-ordinate.</p> <p>X1 may equal X2, and Y1 = Y2 allowing allowing the function to be used for rounded panels, rounded buttons, circular buttons.</p> <p>Rendering can be obtained with <code>gfx_FillPattern(PATTRN)</code>; or <code>gfx_FillPattern(OFF)</code>; for no fill pattern.</p>	
Example	<pre>//Draw graduated colour rounded rectangle gfx_RoundGradient(GRAD_WAVE_HOR, 10, 10, 230, 160, 10, BLACK, WHITE);</pre>	

2.6.53. gfx_PieSlice(cx, cy, spread, radius, step, startangle, endangle, mode)

Syntax	<code>gfx_Arc(cx, cy, spread, radius, step, startangle, endangle, mode);</code>	
Arguments	<code>cx, cy, radius, step, startangle, endangle, mode</code>	
	cx, cy	Center of the slice.
	spread	Center offset: it is used to offset the centrepoint of the pieslice to shift a pie chart piece away from the centrepoint.
	radius	Radius of the Slice.
	step	Step is the stepping angle increment for the fineness of the slice.
	startangle	Starting angle of the slice.
	endangle	Ending angle of the slice.
	mode	mode = 0, no outline. mode = 1, outer circumference line only mode = 2, outer circumference and slice lines.
Returns	Nothing	
Description	Draws a pie slice (filled arc) at xc:yc with radius radius , starting at startangle and ending at endangle . Rendering can be obtained with <code>gfx_FillPattern(PATTRN)</code> ; or <code>gfx_FillPattern(OFF)</code> ; for no fill pattern.	
Example		

2.6.54. gfx_PointWithinBox(x, y, &rect)

Syntax	<code>gfx_PointWithinBox(x, y, &rect);</code>	
Arguments	<code>x, y, &rect</code>	
	<code>x, y</code>	Coordinates
	<code>&rect</code>	An array of 4 vars, x1, y1, width, height.
Returns	<code>status</code>	
	<code>status</code>	Returns true if last touch co-ordinates are within the box test area.
Description	Returns true if last touch co-ordinates are within the box test area.	
Example		

2.6.55. `gfx_PointWithinRectangle(x, y, &recta)`

Syntax	<code>gfx_PointWithinRectangle(x, y, &recta);</code>	
Arguments	<code>x, y, &recta</code>	
	<code>x, y</code>	Coordinates
	<code>&recta</code>	An array of 4 vars, x1, y1, width, height.
Returns	<code>status</code>	
	<code>status</code>	Returns true if last touch co-ordinates are within the rectangle test area.
Description	Returns true if last touch co-ordinates are within the rectangle test area.	
Example		

2.6.56. `gfx_ReadBresLine(x1, y1, x2, y2, ptr)`

Syntax	<code>gfx_readBresLine(x1, y1, x2, y2, ptr);</code>	
Arguments	<code>x1, y1, x2, y2, ptr</code>	
	<code>x1, y1</code>	Line mapping start point.
	<code>x2, y2</code>	Line mapping end point.
	<code>ptr</code>	If zero is passed, an array of the required size to map the line is created. If non zero, it is expected that this is a pointer to an array large enough to store each pixel that is read.
Returns	<code>value</code>	
	<code>value</code>	A pointer to the created array, or the users array. In the case of <code>ptr=0</code> (creation of array), if there is insufficient memory to create the array, zero is returned.
Description	<p>Due to the fact that most LCD displays are not double buffered, and memory is limited on small platforms, <code>gfx_ReadBresLine</code> offers a simple but powerful way of manipulating raster lines by storing all the pixels for an arbitrary line.</p> <p>Typically, <code>gfx_ReadBresLine</code> is used when 'rubber banding' a rectangular area when dragging a marker rectangle, or drawing a needle on a pre- rendered meter or gauge image. The power of this function is further extended when used with the array math functions.</p> <p><code>gfx_ReadBresLine</code> reads an arbitrary line from the display to an array. If "ptr" is 0, the correctly sized array is created, in which case it is up to the caller to eventually destroy it when no longer required. Otherwise "ptr" is expected to point to a correctly sized array.</p> <p>Note: if an array is supplied, its size must be large enough, and may be calculated:- <code>bytecount := (MAX(ABS(x2-x1), ABS(y2-y1) + 1) * 2;</code> <code>// calc array size for mem_Alloc (which allocates byte storage)</code></p> <p><code>wordcount := (MAX(ABS(x2-x1), ABS(y2-y1) + 1);</code> <code>// calc array size for fixed word array (it's much easier to let the function to do this calculation for you – if applicable)</code></p>	
Example		

2.6.57. gfx_WriteBresLine(x1, y1, x2, y2, ptr)

Syntax	<code>gfx_WriteBresLine(x1, y1, x2, y2, ptr);</code>	
Arguments	<code>x1, y1, x2, y2, ptr</code>	
	<code>x1, y1</code>	Line mapping start point.
	<code>x2, y2</code>	Line mapping end point.
	<code>ptr</code>	Points to the array to be written
Returns	Nothing	
Description	Cast pixel values from array to arbitrary line.	
Example		

2.6.58. `gfx_ReadGRAMarea(x1, y1, x2, y2, ptr)`

Syntax	<code>gfx_ReadGRAMarea(x1, y1, x2, y2, ptr);</code>	
Arguments	<code>x1, y1, x2, y2, ptr</code>	
	<code>x1, y1</code>	Top left corner of the rectangular area.
	<code>x2, y2</code>	Bottom right corner of the rectangular area.
	<code>ptr</code>	If zero is passed, an array of the required size to map the line is created. If non zero, it is expected that this is a pointer to an array large enough to store each pixel that is read.
Returns	<code>value</code>	
	<code>value</code>	A pointer to the created array, or the users array. In the case of <code>ptr=0</code> , if there is insufficient memory to create the array, zero is returned.
Description	<p>Reads an arbitrary rectangular area from the display to an array. If "ptr" is 0, the correctly sized array is created, in which case it is up to the caller to eventually destroy it. Otherwise "ptr" is expected to point to a correctly sized array.</p> <p>Note: If an array is supplied, its size must be large enough, and may be calculated:-</p> <p><code>bytecount := (ABS(x2-x1)+1) * (ABS(y2-y1) + 1) * 2; // calc array size for mem_Alloc (which allocates byte storage)</code></p> <p><code>wordcount := (ABS(x2-x1)+1) * ABS(y2-y1); // calc array size for fixed word array</code></p>	
Example		

2.6.59. gfx_WriteGRAMarea(x1, y1, x2, y2, ptr)

Syntax	<code>gfx_WriteGRAMarea(x1, y1, x2, y2, ptr);</code>	
Arguments	<code>x1, y1, x2, y2, ptr</code>	
	<code>x1, y1</code>	Top left corner of the rectangular area.
	<code>x2, y2</code>	Bottom right corner of the rectangular area.
	<code>ptr</code>	Points to an array to be written.
Returns	Nothing	
Description	Write an array back to the rectangular area	
Example		

2.6.60. `gfx_Surround(x1, y1, x2, y2, rad1, rad2, oct, colour)`

Syntax	<code>gfx_Surround(x1, y1, x2, y2, rad1, rad2, oct, color);</code>	
Arguments	<code>x1, y1, x2, y2, rad1, rad2, oct, color</code>	
	<code>x1, y1</code>	Specifies the top left corner position of the surround on the screen.
	<code>x2, y2</code>	Specifies the bottom right corner position of the surround on the screen.
	<code>rad1</code>	Inner corner radius.
	<code>rad2</code>	Outer corner radius.
	<code>oct</code>	0=draw full surround, else control which segments are drawn.
	<code>color</code>	The colour of the surround.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Draws an outline rectangle at the given co-ordinates with optional rounded corners determined by '<code>rad1</code>'.</p> <p>'<code>rad2</code>' is added to '<code>rad1</code>' to form the outer rounded rectangle.</p> <p>If '<code>rad1</code>' is zero, the inner rectangle will have square corners.</p> <p>If '<code>oct</code>' is non zero, various segments may be turned on or off with the given constants. (see "quadrant control constants for rounded objects")</p>	
Example	<pre>gfx_Surround(40, 40, 100, 60, 15, 3, 0, YELLOW);</pre> <p>Draw a surround with rounded corners, 3 pixels wide</p>	

2.6.61. gfx_Scope(Left, Width, Yzero, n, Xstep, Yamp, Colourbg, old_y1, new_y1, Colour1, ... old_y4, new_y4, Colour4)

Syntax	<code>gfx_Scope(left, width, Yzero, N, Xstep, Yamp, colourbg, old_y1, new_y1, colour1, old_y2, new_y2, colour2, old_y3, new_y3, colour3, old_y4, new_y4, colour4);</code>	
Arguments	<code>left, width, Yzero, N, Xstep, Yamp, colourbg, old_y1, new_y1, colour1, old_y2, new_y2, colour2, old_y3, new_y3, colour3, old_y4, new_y4, colour4</code>	
	Left	The left margin of the Scope.
	Width	The width of the Scope.
	Yzero	The y position that corresponds to a y value of zero, normally "Top" + "Height" for a graph, or "Top" + "Height"/2 for a scope.
	N	The number of elements in each buffer. This will need to be greater than "width" for negative "Xstep" values.
	Xstep	X position is incremented each point by "xstep" pixels.
	Yamp	Amplification in the Y direction, 100 is unity.
	ColourBg	The color of the Scope's Background.
	oldy1..4	Buffer containing most recent set of points to be un-drawn.
	newy1..4	Buffer containing new points to be drawn.
	Colour1..4	Colour of the waveform.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Draws up to 4 waveforms from table(s) of vertices at the specified origin. Also useful for drawing line graphs.</p> <p>X position is incremented each point by "Xstep" pixels, values are skipped for negative values.</p> <p>Y values are derived from a Y buffer.</p> <p>After the waveform is drawn, "newy" buffer is automatically copied to "oldy" buffer. Use 0 as the buffers for any unused waveforms.</p>	
Example		

2.7. Display I/O Functions

These functions allow direct display access for fast blitting operations.

Summary of Functions in this section:

- disp_SetReg(register, data)
- disp_setGRAM(x1, y1, x2, y2)
- disp_WrGRAM(colour)
- disp_WriteControl(value)
- disp_WriteWord(value)
- disp_ReadWord()
- disp_Disconnect()
- disp_Init()

2.7.1. disp_SetReg(register, data)

Syntax	<code>disp_SetReg(register, data);</code>	
Arguments	<code>register, data</code>	
	<code>register</code>	Refer to the display driver data sheet
	<code>data</code>	Refer to the display driver data sheet
Returns	<code>nothing</code>	
Description	Sets the Display driver IC register.	

2.7.2. disp_setGRAM(x1, y1, x2, y2)

Syntax	disp_setGRAM(x1, y1, x2, y2);	
Arguments	x1, y1, x2, y2	
	x1, y1	Top left of the GRAM window.
	x2, y2	Bottom right of the GRAM window.
Returns	value	
	value	The LO word of the 32 bit pixel count is returned.
Description	Prepares the GRAM area for user access. The lower 16bits of the pixel count in the selected area is returned. This is usually all that is needed unless GRAM area exceeds 256 ² . A copy of the 32bit value can be found in GRAM_PIXEL_COUNT_LO and GRAM_PIXEL_COUNT_HI.	

2.7.3. disp_WrGRAM(colour)

Syntax	disp_WrGRAM(colour);	
Arguments	colour	
	colour	Pixel color to be populated.
Returns	nothing	
Description	Data can be written to the GRAM consecutively using this function once the GRAM access window has been setup.	

2.7.4. disp_WriteControl(value)

Syntax	<code>disp_WriteControl(value);</code>	
Arguments	value	
	value	Specifies the 16 bit value to be written to the display control register.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Sends a 16 bit value to the display bus. Refer to individual data sheets for the display for more information. This function is used to extend the capabilities of the user code to gain access to the display hardware.	

2.7.5. disp_WriteWord(value)

Syntax	<code>disp_WriteWord(value);</code>	
Arguments	value	
	value	Specifies the value to be written to the display data register.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Sends a 16 bit value to the display bus. Refer to individual data sheets for the display for more information. This function is used to extend the capabilities of the user code to gain access to the the display hardware.	

2.7.6. disp_ReadWord(value)

Syntax	<code>disp_ReadWord();</code>	
Arguments	nothing	
Returns	value	
	value	Returns 16 bit value in the register.
Description	Read a word from the display.	

2.7.7. disp_Disconnect()

Syntax	<code>disp_Disconnect();</code>
Arguments	none
Returns	nothing
Description	<p>This function disconnects the display driver pins and/or reconfigures it to achieve its lowest possible power consumption. Use after disabling peripheral power to ensure the minimal power usage by the display. <code>Disp_Init()</code> should be used to reinitialise the display.</p> <p>New in v0.7 PmmC</p>

2.7.8. disp_Init()

Syntax	<code>disp_Init();</code>
Arguments	none
Returns	nothing
Description	<p>This function is used to initialise the display.</p> <p>This is useful in a number of situations, however mainly for the uLCD-xx-PTU modules which have the ability to disable the power supply to the display for low power sleep modes. This function is required to re-initialise the display once power to the display has been restored, so the display is usable once again.</p> <p>New in v0.7 PmmC</p>

2.7.9. disp_BlitPixelsFromCOM0() or disp_BlitPixelsFromCOM1() or disp_BlitPixelsFromCOM2() or disp_BlitPixelsFromCOM3()

Syntax	<code>disp_BlitPixelsFromCOM0;</code> or <code>disp_BlitPixelsFromCOM1;</code> or <code>disp_BlitPixelsFromCOM2;</code> or <code>disp_BlitPixelsFromCOM3</code>
Arguments	None
Returns	Nothing
Description	This function writes the number of pixels defined by the last <code>disp_setGRAM()</code> call to the display from the specified com port. The function returns once all pixels have been written. New in v1.1 PmmC

2.8. Media Functions (SD/SDHC Memory Card or Serial Flash chip)

The media can be SD/SDHC, microSD or serial (NAND) flash device interfaced to the DIABLO16 SPI port.

Summary of Functions in this section:

- `media_Init()`
- `media_SetAdd(HIword, LOword)`
- `media_SetSector(HIword, LOword)`
- `media_RdSector(Destination_Address)`
- `media_WrSector(Source_Address)`
- `media_ReadByte()`
- `media_ReadWord()`
- `media_WriteByte(byte_val)`
- `media_WriteWord(word_val)`
- `media_Flush()`
- `media_Image(x, y)`
- `media_Video(x, y)`
- `media_VideoFrame(x, y, frameNumber)`

2.8.1. media_Init()

Syntax	<code>media_Init();</code>	
Arguments	none	
Returns	result	
	result	Returns: 1 if memory card is present and successfully initialised Returns: 0 if no card is present or not able to initialise
Description	Initialise a uSD/SD/SDHC memory card for further operations. The SD card is connected to the SPI (serial peripheral interface) of the processor.	
Example	<pre>while(!media_Init()) gfx_Cls(); pause(300); puts("Please insert SD card"); pause(300); wend</pre> <p>This example waits for SD card to be inserted and initialised, flashing a message if no SD card detected.</p>	

2.8.2. media_SetAdd(HIword, LOword)

Syntax	<code>media_SetAdd(HIword, LOword);</code>	
Arguments	HIword, LOword	
	HIword	Specifies the high word (upper 2 bytes) of a 4 byte media memory byte address location.
	LOword	Specifies the low word (lower 2 bytes) of a 4 byte media memory byte address location.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Set media memory internal Address pointer for access at a non sector aligned byte address.	
Example	<pre>media_SetAdd(0, 513);</pre> <p>This example sets the media address to byte 513 (which is sector #1, 2nd byte in sector) for subsequent operations.</p>	

2.8.3. media_SetSector(HIword, LOword)

Syntax	<code>media_SetSector(HIword, LOword);</code>	
Arguments	HIword, LOword	
	HIword	Specifies the high word (upper 2 bytes) of a 4 byte media memory sector address location.
	LOword	Specifies the low word (lower 2 bytes) of a 4 byte media memory sector address location.
	The arguments can be a variable, array element, expression or constant	
Returns	result	
Description	Set media memory internal Address pointer for sector access.	
Example	<pre>media_SetSector(0, 10);</pre> <p>This example sets the media address to the 11th sector (which is also byte address 5120) for subsequent operations</p>	

2.8.4. media_RdSector(Destination_Address)

Syntax	<code>media_RdSector(Destination_Address);</code>	
Arguments	Destination_Address	
	Destination_Address	Destination block pointed to by the internal Sector pointer. The argument must be a pointer to an array of size 256 words for the sector data which will be 512 bytes
Returns	Returns TRUE if media response was TRUE. Returns 512 bytes (256 words) in to a destination block.	
Description	Reads and Returns 512 bytes (256 words) into a destination block (eg <code>rdblock[256]</code>) pointed to by the internal Sector pointer. After the read the Sector pointer is automatically incremented by 1.	
Example	<pre>var rdblock[256]; media_SetSector(0,10) if (media_RdSector(rdblock)); Print("Data collected"); endif</pre> <p>This example sets a 512 bytes block and collects data from the address pointed to by <code>media_SetSector</code> command.</p>	

2.8.5. media_WrSector(Source_Address)

Syntax	<code>media_WrSector(Source_Address);</code>	
Arguments	Source_Address	
	Source_Address	Source memory block of 512bytes.
	The arguments can be a variable, array element, expression or constant	
Returns	Returns TRUE if media response was TRUE.	
Description	Writes 512 bytes (256 words) from a source memory block (eg wrblock[256]) into the uSD card. After the write the Sect pointer is automatically incremented by 1. Returns TRUE if uSD response was TRUE	
Example	<pre>var wrblock[256]; func main() prepare_block(); media_SetSector(0,10) if (media_WrSector(wrblock)); Print("Data transferred"); endif : : This example sets a 512 bytes block and transfers data to the address pointed to by media_SetSector command.</pre>	

2.8.6. media_ReadByte()

Syntax	<code>media_ReadByte();</code>
Arguments	none
Returns	byte value
Description	Returns the byte value from the current media address. The internal byte address will then be internally incremented by one.
Example	<pre>var LObyte, HIbyte; if(media_Init()) media_SetAdd(0, 510); LObyte := media_ReadByte(); HIbyte := media_ReadByte(); print([HEX2]HIbyte, [HEX2]LObyte); endif repeat forever</pre> <p>This example initialises the media, sets the media byte address to 510, and reads the last 2 bytes from sector 0. If the card happens to be FAT formatted, the result will be "AA55". The media internal address is internally incremented for each of the byte operations.</p>

2.8.7. media_ReadWord()

Syntax	<code>media_ReadWord();</code>
Arguments	none
Returns	word value
Description	Returns the word value (2 bytes) from the current media address. The internal byte address will then be internally incremented by two. If the address is not aligned, the word will still be read correctly.
Example	<pre>var myword; if(media_Init()) media_SetAdd(0, 510); myword := media_ReadWord(); print([HEX4]myword); endif repeat forever</pre> <p>This example initialises the media, sets the media byte address to 510 and reads the last word from sector 0. If the card happens to be formatted, the result will be "AA55"</p>

2.8.8. media_WriteByte(byte_val)

Syntax	media_WriteByte(byte_val);	
Arguments	byte_val	
	byte_val	The lower 8 bits specifies the byte to be written at the current media address location.
	The arguments can be a variable, array element, expression or constant	
Returns	success	
	success	Returns non zero if write was successful.
Description	Writes a byte to the current media address that was initially set with media_SetAdd() or media_SetSector(...) ; After the write the Address pointer is automatically incremented by 1. Note: Writing bytes or words to a media sector must start from the beginning of the sector. All writes will be incremental until the media_Flush() function is executed, or the sector address rolls over to the next sector. When media_Flush() is called, any remaining bytes in the sector will be padded with 0xFF , destroying the previous contents. An attempt to use the media_SetAdd(..) function will result in the lower 9 bits being interpreted as zero. If the writing rolls over to the next sector, the media_Flush() function is issued automatically internally.	
Example	<pre> var n, char; while (media_Init()==0); // wait if no SD card detected media_SetSector(0, 2); // at sector 2 //media_SetAdd(0, 1024); // (alternatively, use media_SetAdd(), // lower 9 bits ignored) while (n < 10) media_WriteByte(n++ +'0'); // write ASCII '0123456789' to the wend // first 10 locations. to(MDA); putstr("Hello World"); // now write a ascii test string media_WriteByte('A'); // write a further 3 bytes media_WriteByte('B'); media_WriteByte('C'); media_WriteByte(0); // terminate with zero media_Flush(); // we're finished, close the sector media_SetAdd(0, 1024+5); // set the starting byte address while(char:=media_ReadByte()) putch(char); // print result, starting // from '5' repeat forever </pre> <p>This example initialises the media, writes some bytes to the required sector, then prints the result from the required location.</p>	

2.8.9. media_WriteWord(word_val)

Syntax	media_WriteWord(word_val);	
Arguments	word_val	
	word_val	The 16 bit word to be written at the current media address location.
	The arguments can be a variable, array element, expression or constant	
Returns	success	
	success	Returns non zero if write was successful.
Description	Writes a word to the current media address that was initially set with media_SetAdd() or media_SetSector(...) ; After the write the Address pointer is automatically incremented by 2. Note: Writing bytes or words to a media sector must start from the beginning of the sector. All writes will be incremental until the media_Flush() function is executed, or the sector address rolls over to the next sector. When media_Flush() is called, any remaining bytes in the sector will be padded with 0xFF, destroying the previous contents. An attempt to use the media_SetAdd(..) function will result in the lower 9 bits being interpreted as zero. If the writing rolls over to the next sector, the media_Flush() function is issued automatically internally.	
Example	<pre> var n; while (media_Init()==0); // wait until a good SD card is found n:=0; media_SetAdd(0, 1536); // set the starting byte address while (n++ < 20) media_WriteWord(RAND()); // write 20 random words to first 20 wend // word locations. n:=0; while (n++ < 20) media_WriteWord(n++*1000); // write sequence of 1000*n to next 20 wend // word locations. media_Flush(); // we're finished, close the sector media_SetAdd(0, 1536+40); // set the starting byte address n:=0; while(n++<8) // print result of fist 8 multiplication calcs print([HEX4] media_ReadWord(),"\n"); wend repeat forever // This example initialises the media, writes some words to the required sector, then prints // the result from the required location. </pre>	

2.8.10. media_Flush()

Syntax	<code>media_Flush();</code>
Arguments	none
Returns	returns 0 if Failed returns non-zero if OK
Description	After writing any data to a sector, <code>media_Flush()</code> should be called to ensure that the current sector that is being written is correctly stored back to the media else write operations may be unpredictable.
Example	See the media_WriteByte(..) and media_WriteWord(..) examples.

2.8.11. `media_Image(x, y)`

Syntax	<code>media_Image(x, y);</code>	
Arguments	<code>x, y</code>	
	<code>x, y</code>	Specifies the top left position where the image will be displayed.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Displays an image from the media storage at the specified co-ordinates. The image address is previously specified with the <code>media_SetAdd(..)</code> or <code>media_SetSector(...)</code> function. If the image is shown partially off screen, it may not be displayed correctly.	
Example	<pre>while(media_Init()==0); // wait if no SD card detected media_SetAdd(0x0001, 0xDA00); // point to the books04 image media_Image(10,10); gfx_Clippping(ON); // turn off clipping to see the difference media_Image(-12,50); // show image off-screen to the left media_Image(50,-12); // show image off-screen at the top repeat forever</pre> <p>This example draws an image at several positions, showing the effects of clipping.</p>	

2.8.12. media_Video(x, y)

Syntax	<code>media_Video(x, y);</code>	
Arguments	<code>x, y</code>	
	<code>x, y</code>	Specifies the top left position where the video clip will be displayed.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Displays a video clip from the media storage device at the specified co-ordinates. The video address location in the media is previously specified with the media_SetAdd(..) or media_SetSector(...) function. If the video is shown partially off screen, it may not be displayed correctly. Note that showing a video blocks all other processes until the video has finished showing. See the media_VideoFrame(...) functions for alternatives.	
Example	<pre>while(media_Init()==0); // wait if no SD card detected media_SetAdd(0x0001, 0x3C00); // point to the 10-gear clip media_Video(10,10); gfx_Clipping(ON); // turn off clipping to see the difference media_Video(-12,50); // show video off-screen to the left media_Video(50,-12); // show video off-screen at the top repeat forever</pre> <p>This example plays a video clip at several positions, showing the effects of clipping.</p>	

2.8.13. media_VideoFrame(x, y, frameNumber)

Syntax	media_VideoFrame(x, y, frameNumber);	
Arguments	x, y	
	x, y	Specifies the top left position where the video clip will be displayed.
	frameNumber	Specifies the required frame to be shown.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>Displays a video from the media storage device at the specified co-ordinates. The video address is previously specified with the media_SetAdd(..) or media_SetSector(...) function. If the video is shown partially off it may not be displayed correctly. The frames can be shown in any order. This function gives you great flexibility for showing various icons from an image strip, as well as showing videos while doing other tasks</p> <p>media_VideoFrame(..) will now show error box for out of range video frames. Also, if frame is set to -1, just a rectangle will be drawn in background colour to blank an image. It applies to PmmC R29 or above.</p>	
Example	<pre>var frame; while (media_Init()==0); // wait if no SD card detected while (media_Init()==0); // wait if no SD card detected media_SetAdd(0x0002, 0x3C00); // point to the 10-gear image repeat frame := 0; // start at frame 0 repeat media_VideoFrame(30,30, frame++); // display a frame pause(peekB(IMAGE_DELAY)); // pause for the time given in // the image header until(frame == peekW(IMG_FRAME_COUNT)); // loop until we've // shown all the frames forever // do it forever</pre> <p>This first example shows how to display frames as required while possibly doing other tasks. Note that the frame timing (although not noticeable in this small example) is not correct as the delay commences after the image frame is shown, therefore adding the display overheads to the frame delay. This second example employs a timer for the framing delay, and shows the same movie simultaneously running forward and backwards with time left for other tasks as well. A number of videos (or animated icons) can be shown simultaneously using this method.</p> <pre>var framecount, frame, delay, colr; frame := 0; // show the first frame so we can get the video header info // into the system variables, and then to our local variables. media_VideoFrame(30,30, 0); framecount := peekW(IMG_FRAME_COUNT); // we can now set some local // values. delay := peekB(IMAGE_DELAY); // get the frame count and delay repeat repeat pokeW(TIMER0, delay); // set a timer</pre>	

```
media_VideoFrame(30,30, frame++); // show next frame
gfx_MoveTo(64,35);
print([DEC2Z] frame); // print the frame number
media_VideoFrame(30,80, framecount-frame); // show movie
// backwards.
gfx_MoveTo(64,85);
print([DEC2Z] framecount-frame); // print the frame number

if ((frame & 3) == 0)
    gfx_CircleFilled(80,20,2,colr); // a blinking circle fun
    colr := colr ^ 0xF800; // alternate colour,
endif // BLACK/RED using XOR
// do more here if required
while(peekW(TIMER0)); // wait for timer to expire
until(frame == peekW(IMG_FRAME_COUNT));
frame := 0;
forever
```

2.9. Flash Memory Chip Functions

The functions in this section apply to the Flash Memory Banks on the DIABLO16.

Summary of Functions in this section:

- flash_Bank()
- flash_Blit1(bank, offset, count, pallete2colour)
- flash_Blit16(bank, offset, count)
- flash_Blit2(bank, offset, count, pallete4colour)
- flash_Blit4(bank, offset, count, pallete16colour)
- flash_Blit8(bank, offset, count)
- flash_Copy(bank, ptr, dest, count)
- flash_EraseBank(bank, confirmation)
- flash_Exec(bank, arglistptr)
- flash_GetByte(bank, ptr)
- flash_GetWord(bank, ptr)
- flash_LoadFile(bank, filename)
- flash_putstr(bank, ptr)
- flash_Run(bank)
- flash_WriteBlock(sourceptr, bank, page)

2.9.1. flash_Bank()

Syntax	<code>flash_Bank();</code>	
Arguments	none	
Returns	value	The FLASH bank that code is currently running from, 0-5. 0: Flashbank 0 1: Flashbank 1 2: Flashbank 2 3: Flashbank 3 4: Flashbank 4 5: Flashbank 5
Description	Identifies which flash bank the code is running from.	

2.9.2. flash_Blit1(bank, offset, count, pallete2colour)

Syntax	flash_Blit1(bank, offset, count, pallete2colour)	
Arguments	bank, offset, count, pallete2colour	
	bank	Flash bank to load the image from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	offset	Offset in to the Flash bank where image is stored.
	count	Total number of pixel in the image.
	pallete2colour	An array of 2 elements being the colors for the two possible colour values.
Returns	count	
	count	Actual count (normally same as count, will be lower if bank bounds exceeded)
Description	Blit an image to a GRAM window from FLASH storage. Image is stored in a linear fashion to suit the GRAM mechanism, palette is 2 x 16bit colours	

2.9.3. flash_Blit2(bank, offset, count, pallete4colour)

Syntax	flash_Blit2(bank, offset, count, pallete4colour)	
Arguments	bank, offset, count, pallete4colour	
	bank	Flash bank to load the image from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	offset	Offset in to the Flash bank where image is stored.
	count	Total number of pixel in the image.
	pallete4colour	An array of 4 elements being the colors for the four possible colour values.
Returns	count	
	count	Actual count (normally same as count, will be lower if bank bounds exceeded)
Description	Blit an image to a GRAM window from FLASH storage. Image is stored in a linear fashion to suit the GRAM mechanism, palette is 4 x 16bit colours	

2.9.4. flash_Blit4(bank, offset, count, pallete16colour)

Syntax	flash_Blit4(bank, offset, count, pallete16colour)	
Arguments	bank, offset, count, pallete16colour	
	bank	Flash bank to load the image from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	offset	Offset in to the Flash bank where image is stored.
	count	Total number of pixel in the image.
	Pallete16colour	An array of 16 elements being the colors for the sixteen possible colour values.
Returns	count	
	count	Actual count (normally same as count, will be lower if bank bounds exceeded)
Description	Blit an image to a GRAM window from FLASH storage. Image is stored in a linear fashion to suit the GRAM mechanism, palette is 16 x 16bit colours	

2.9.5. flash_Blit8(bank, offset, count)

Syntax	flash_Blit8(bank, offset, count)	
Arguments	bank, offset, count	
	bank	Flash bank to load the image from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	offset	Offset in to the Flash bank where image is stored.
	count	Total number of pixel in the image.
Returns	count	
	count	Actual count (normally same as count, will be lower if bank bounds exceeded)
Description	Blit an image to a GRAM window from FLASH storage. Image is stored 8 bits per pixel (332 format) in a linear fashion to suit the GRAM mechanism	

2.9.6. flash_Blit16(bank, offset, count)

Syntax	flash_Blit16(bank, offset, count)	
Arguments	bank, offset, count	
	bank	Flash bank to load the image from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	offset	Offset in to the Flash bank where image is stored.
	count	Total number of pixel in the image.
Returns	count	
	count	Actual count (normally same as count, will be lower if bank bounds exceeded)
Description	Blit an image to a GRAM window from FLASH storage. Image is stored 16bits per pixel (565) in a linear fashion to suit the GRAM mechanism	

2.9.7. flash_Copy(bank, ptr, dest, count)

Syntax	flash_Copy(bank, ptr, dest, count)	
Arguments	bank, ptr, dest, count	
	bank	Flash bank to copy the data from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	ptr	Pointer to a location in the selected flash bank.
	dest	Pointer to the destination. The destination pointer is byte aligned, so a str_Ptr must be raised to get correct address.
	count	Count of bytes to be transferred.
Returns	count	
	count	The count of bytes transferred.
Description	Copies bytes from any flash locations to a user buffer. If the bank is read protected, 0 bytes will be read.	

2.9.8. flash_EraseBank(bank, confirmation)

Syntax	flash_EraseBank(bank, confirmation)	
Arguments	bank, confirmation	
	bank	Flash bank to be erased. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5 -1 or ALL to select all the banks.
	confirmation	0xDEAD: The command will erase regardless of FLASH_WRITE_PROTECT status For any other value, a protected bank will not be erased.
Returns	status	
	status	Returns true if the function succeeded.
Description	<p>This function should be used with extreme caution. The selected bank will be completely erased regardless of FLASH_WRITE_PROTECT status if the confirmation value is set to hex 0xDEAD. If confirmation is any other value, a protected bank will not be erased, and function will return with 0. If the destination bank is the same as the execution bank, the processor will reset upon completion of erase. If the "bank" argument is set to ALL (-1) and confirmation is set to 0xDEAD, FLASHBANK_0 thru FLASHBANK_5 are cleared.</p> <p>Note: Use with caution, this is a good way to 'clean up' the entire flash when starting new projects.</p> <p>Note: reset processor if program is erasing itself, or the ALL bank option is selected.</p>	

2.9.9. flash_Exec(flashbank, arglistptr)

Syntax	<code>flash_Exec(flashbank, arglistptr);</code>	
Arguments	flashbank, arglistptr	
	flashbank	name of the bank to be executed.
	arglistptr	pointer to the list of arguments to pass to the selected bank or 0 if no arguments.
Returns	Value	
	Value	Returns the value from main in the called bank.
Description	<p>This function calls the main function in another bank. The main program in FLASH retains all memory allocations (eg file buffers, memory allocated with mem_Alloc etc)</p> <p>The called bank returns like a function, program in current bank is kept active and control returns to it. All memory allocated in the called bank should be freed before returning, or it will be lost.</p> <p>If arglistptr is 0, no arguments are passed, else arglist points to an array, the first element being the number of elements in the array.</p> <p>func 'main' in the called bank accepts the arguments.</p>	
Example	<code>flash_Exec(FLASHBANK_1, 0) ;</code>	

2.9.10. flash_GetByte(bank, ptr)

Syntax	<code>flash_GetByte(bank, ptr)</code>
Arguments	bank, ptr

	bank	Flash bank to get the byte from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	ptr	Pointer to a location in the selected flash bank.
Returns	byte	
	byte	The byte value from the location
Description	Reads a single byte from any flash location. If the bank is read protected, only the first 2 bytes can be read. 0x55, 0xAA are the header signature bytes of a valid program.	

2.9.11. flash_GetWord(bank, ptr)

Syntax	flash_GetWord(bank, ptr)
Arguments	bank, ptr

	bank	Flash bank to get the word from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	ptr	Pointer to a location in the selected flash bank.
Returns	word	
	word	The word value from the location
Description	Reads a single word from any flash location. If the bank is read protected, only the first word can be read. 0x55AA is the header signature word of a valid program.	

2.9.12. flash_LoadFile(bank, filename)

Syntax	flash_LoadFile (bank, filename)
Arguments	bank, filename

	bank	Flash bank to load the file from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	filename	Name of the file to be copied (passed as a string).
Returns	status	
	status	Returns true if the function succeeded
Description	Copies a file from uSD to the required flashbank. The destination bank cannot be the execution bank, or a bank that is write protected.	

2.9.13. flash_putstr(bank, ptr)

Syntax	flash_putstr(bank, ptr)	
Arguments	bank, ptr	
	bank	Flash bank to load the String from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	ptr	Pointer to a NULL terminated string in the selected flash bank.
Returns	status	
	status	True if function succeeds, usually ignored. 0 if bank is read protected.
Description	Prints an ASCIIZ string from the Flash bank. Works the same as putstr, however, the source of the ASCIIZ string is in FLASH storage. Output may be redirected with the to(..) function. Bit15 of ptr is assumed 0.	

2.9.14. flash_Run(bank)

Syntax	flash_Run(bank)	
Arguments	bank	
	bank	Flash bank to load the program from. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
Returns	value	This function should not return as it restarts the processor and jumps to the required bank. If it does return, -1 indicates incorrect/invalid bank number. -2 indicates no valid program in the selected bank.
Description	Restarts the processor, running code from the required flash bank. Bank may be a variable, or one of the pre-defined constants.	

2.9.15. flash_WriteBlock(sourceptr, bank, page)

Syntax	flash_WriteBlock(sourceptr, bank, page)	
Arguments	sourceptr, bank, page	
	sourceptr	Source buffer to load the 2K bytes of data from.
	bank	Flash bank to write the block to. 0 or FLASHBANK_0 1 or FLASHBANK_1 2 or FLASHBANK_2 3 or FLASHBANK_3 4 or FLASHBANK_4 5 or FLASHBANK_5
	page	Page number 0-15. Each page is 2K. The address of each block is 0, 2048, 4096 etc, determined by the page number 0-15.
Returns	status	
	status	Returns true if the function succeeded.
Description	Copies a 2kbyte buffer to the required flashbank in block 0-15. The destination bank cannot be an execution bank, or a program bank that is write-protected.	

2.10. SPI Control Functions

The SPI functions in this section apply to any general purpose SPI device.

Summary of Functions in this section:

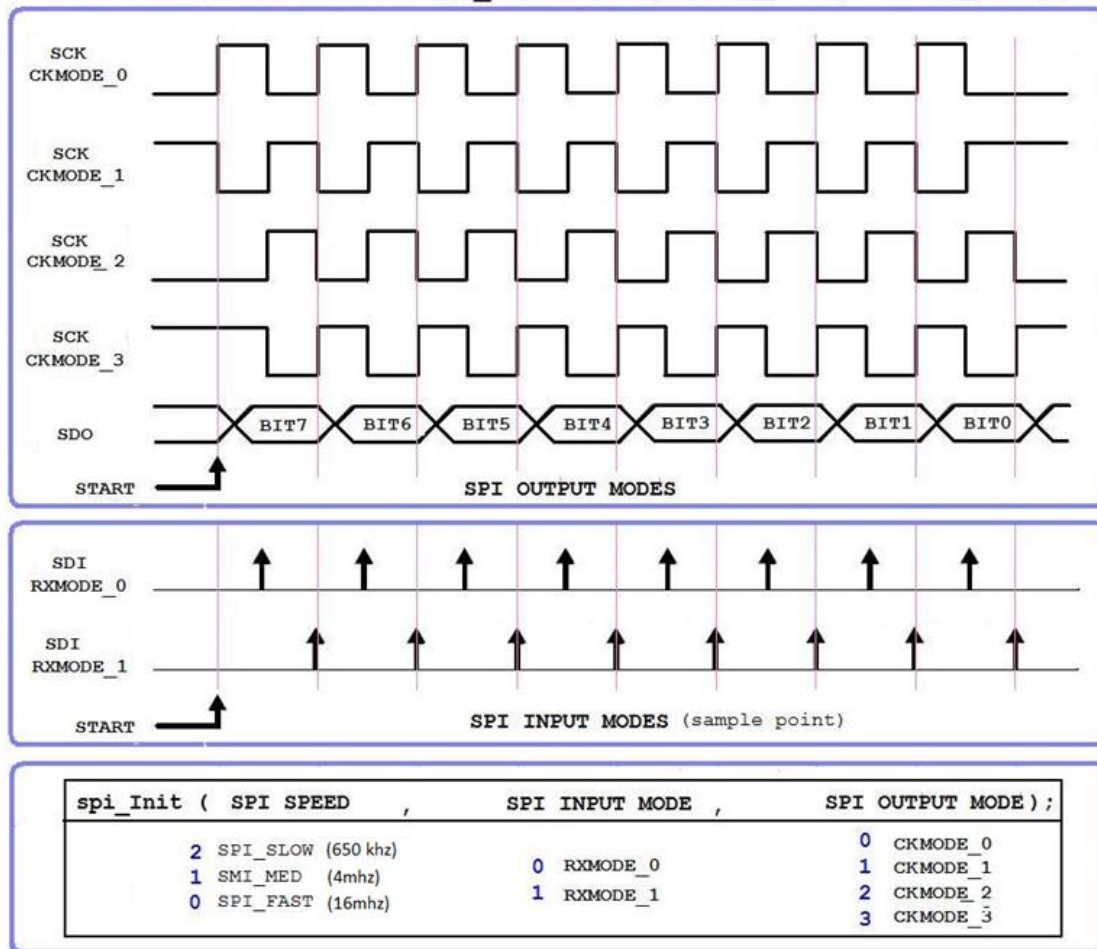
- spi_Init(speed, input_mode, output_mode)
- spi_Read()
- spi_Write(byte)
- spi_Disable()
- SPI1_Init(speed, mode) **or** SPI2_Init(speed, mode) **or** SPI3_Init(speed, mode)
- SPI1_Read() **or** SPI2_Read() **or** SPI3_Read()
- SPI1_Write(byte) **or** SPI2_Write(byte) **or** SPI3_Write(byte)
- SPI1_SCK_pin(pin) **or** SPI2_SCK_pin(pin) **or** SPI3_SCK_pin(pin)
- SPI1_SDI_pin(pin) **or** SPI2_SDI_pin(pin) **or** SPI3_SDI_pin(pin)
- SPI1_SDO_pin(pin) **or** SPI2_SDO_pin(pin) **or** SPI3_SDO_pin(pin)

Note: SPI0 is connected internally to the uSD card. spi_Init(), spi_Read(), spi_Write() and spi_Disable() all refer to the SPI0 to communicate with the uSD card through direct SPI commands. Only adept users should attempt this as it might damage the uSD card.

2.10.1. spi_Init(speed, input_mode, output_mode)

Syntax	<code>spi_Init(speed, input_mode, output_mode);</code>	
Arguments	<code>speed, input_mode, output_mode</code>	
	speed	Sets the speed of the SPI port.
	input_mode	Sets the input mode of the SPI port. See diagram below.
	output_mode	Sets the output mode of the SPI port. See diagram below.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Sets up the DIABLO SPI port to communicate with the uSD card through direct SPI commands. It should not be used if uSD card is active. Note: This is only for the uSD Card, it is <u>not</u> for SPI1, SPI2 or SPI3 WARNING: This should not be tampered with for normal operation, as the DIABLO16 handles the uSD card itself. Only use if you are an adept user and know what you are doing.	

SPI MODE ARGUMENTS FOR `spi_Init(SPEED, INPUT_MODE, OUTPUT_MODE);`



2.10.2. spi_Read()

Syntax	<code>spi_Read();</code>	
Arguments	none	
Returns	byte	
	byte	Returns a single data byte from the uSD card via SPI.
Description	This function allows a raw unadorned byte read from the uSD card via SPI. Note: This is only for the uSD Card, it is <u>not</u> for SPI1, SPI2 or SPI3 Note: The Chip Select line (SDCS) is lowered automatically.	

2.10.3. spi_Write(byte)

Syntax	<code>spi_Write(byte);</code>	
Arguments	byte	
	byte	Specifies the data byte to be sent to the uSD card via SPI.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	This function allows a raw unadorned byte write to the uSD card via SPI. Note: This is only for the uSD Card, it is <u>not</u> for SPI1, SPI2 or SPI3 Note: The Chip Select line (SDCS) is lowered automatically.	

2.10.4. spi_Disable()

Syntax	<code>spi_Disable();</code>
Arguments	none
Returns	nothing
Description	<p>This function raises the Chip Select (SDCS) line of the uSD card, disabling it from further activity. The CS line will be automatically lowered next time the SPI functions <code>spi_Read()</code> or <code>spi_Write(...)</code> are used, and also by action of any of the <code>media_</code> functions.</p> <p>Note: This is only for the uSD Card, it is <u>not</u> for SPI1, SPI2 or SPI3</p>

2.10.5. SPI1_Init(speed, mode) or SPI2_Init(speed, mode) or SPI3_Init(speed, mode)

Syntax	SPI1_Init(speed, mode); or SPI2_Init(speed, mode); or SPI3_Init(speed, mode);	
Arguments	speed, mode	
speed	Specifies the speed of the SPI port. See the details below,	
	mode	Pre-defined Constant
	0	SPI_SPEED0
	1	SPI_SPEED1
	2	SPI_SPEED2
	3	SPI_SPEED3
	4	SPI_SPEED4
	5	SPI_SPEED5
	6	SPI_SPEED6
	7	SPI_SPEED7
	8	SPI_SPEED8
	9	SPI_SPEED9
	10	SPI_SPEED10
	11	SPI_SPEED11
	12	SPI_SPEED12
	13	SPI_SPEED13
	14	SPI_SPEED14
15	SPI_SPEED15	
mode	Specifies the mode of SPI operation. See the details below,	
	mode	Pre-defined Constant
	8 bit Mode	
	0	SPI8_MODE_0
	1	SPI8_MODE_1
	2	SPI8_MODE_2
	3	SPI8_MODE_3
	4	SPI8_MODE_4
	5	SPI8_MODE_5
	6	SPI8_MODE_6
	7	SPI8_MODE_7
	16 bit Mode	
	8	SPI16_MODE_0
	9	SPI16_MODE_1

			SDI sampled on first rising edge
10	SPI16_MODE_2		SCK idles high, SDO stable for first falling edge, SDI sampled on first falling edge
11	SPI16_MODE_3		SCK idles high, SDO stable for first rising edge, SDI sampled on first falling edge
12	SPI16_MODE_4		SCK idles low, SDO stable for first falling edge, SDI sampled on next rising edge
13	SPI16_MODE_5		SCK idles low, SDO stable for first rising edge, SDI sampled on next falling edge
14	SPI16_MODE_6		SCK idles high, SDO stable for first falling edge, SDI sampled on next rising edge
15	SPI16_MODE_7		SCK idles high, SDO stable for first rising edge, SDI sampled on next rising edge
Returns			
	status		
	status	Returns true if the function succeeded.	
Description			
	Initialize the SPI port to communicate with the SPI device. There are three peripheral interfacable SPI ports that can be used to communicate with three different SPI devices with different speeds and modes at the same time. SPI1, SPI2 and SPI3 need to be initialized separately using SPI1_Init(..), SPI2_Init(..) or SPI3_Init(..) functions.		
	Note: This is only for SPI1, SPI2 or SPI3, it is separate from the spi_Init() function used for the uSD Card		

2.10.6. SPI1_Read() or SPI2_Read() or SPI3_Read()

Syntax	SPI1_Read(); or SPI2_Read(); or SPI3_Read();	
Arguments	none	
Returns	byte	
	byte	Returns a single data byte from the SPI device.
Description	<p>This function allows a raw unadorned byte read from the SPI device connected to SPI1, SPI2 or SPI3 port. A dummy write using all bits set is automatically written to the SPI port to being the read.</p> <p>Note: The Chip Select line needs to be manually lowered and raised by the users' code since this pin is determined by the user and is not a fixed pin.</p> <p>Note: This is only for SPI1, SPI2 or SPI3, it is separate from the spi_Read() function used for the uSD Card</p>	

2.10.7. SPI1_Write(byte) or SPI2_Write(byte) or SPI3_Write(byte)

Syntax	SPI1_Write(byte); or SPI2_Write(byte); or SPI3_Write(byte);	
Arguments	byte	
	byte	Specifies the data byte to be sent to the SPI device.
	The arguments can be a variable, array element, expression or constant	
Returns	Data	
	Data	Returns the data read from the SPI port whilst the write is in progress
Description	<p>This function allows a raw unadorned byte write to the SPI device connected to SPI1, SPI2 or SPI3 port.</p> <p>Note: The Chip Select line needs to be manually lowered and raised by the users' code since this pin is determined by the user and is not a fixed pin.</p> <p>Note: This is only for SPI1, SPI2 or SPI3, it is separate from the spi_Write() function used for the uSD Card</p>	

2.10.8. SPI1_SCK_pin(pin) or SPI2_SCK_pin(pin) or SPI3_SCK_pin(pin)

Syntax	SPI1_SCK_pin(pin); or SPI2_SCK_pin(pin); or SPI3_SCK_pin(pin);																																																																						
Arguments	pin																																																																						
	pin	Specifies the pin to be set for SCK for SPI1, SPI2 or SPI3 ports.																																																																					
		<table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>H1 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>1</td><td>No</td></tr> <tr><td>PA1</td><td>62</td><td>3</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>5</td><td>No</td></tr> <tr><td>PA3</td><td>64</td><td>7</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>29</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>27</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>25</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>23</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>21</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>19</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>8</td><td>No</td></tr> <tr><td>PA11</td><td>44</td><td>6</td><td>No</td></tr> <tr><td>PA12</td><td>31</td><td>28</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>30</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>24</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>26</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	H1 Pin Number	Availability	PA0	61	1	No	PA1	62	3	Yes	PA2	63	5	No	PA3	64	7	Yes	PA4	46	29	Yes	PA5	49	27	Yes	PA6	50	25	Yes	PA7	51	23	Yes	PA8	52	21	Yes	PA9	53	19	Yes	PA10	43	8	No	PA11	44	6	No	PA12	31	28	Yes	PA13	32	30	Yes	PA14	37	24	No	PA15	36	26	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	H1 Pin Number	Availability																																																																				
PA0	61	1	No																																																																				
PA1	62	3	Yes																																																																				
PA2	63	5	No																																																																				
PA3	64	7	Yes																																																																				
PA4	46	29	Yes																																																																				
PA5	49	27	Yes																																																																				
PA6	50	25	Yes																																																																				
PA7	51	23	Yes																																																																				
PA8	52	21	Yes																																																																				
PA9	53	19	Yes																																																																				
PA10	43	8	No																																																																				
PA11	44	6	No																																																																				
PA12	31	28	Yes																																																																				
PA13	32	30	Yes																																																																				
PA14	37	24	No																																																																				
PA15	36	26	No																																																																				
Returns	status																																																																						
	status	Returns TRUE if function succeeded (usually ignored)																																																																					
Description	Selects the hardware pin for spi Clock line. SPI1, SPI2 or SPI3's SCK pin could be assigned to the available pins. Note that only a single pin should be mapped to spi SCK. If the pin argument is 0 the previously selected spi SCK pin is disconnected. The pin is automatically set to an output.																																																																						

2.10.9. SPI1_SDI_pin(pin) or SPI2_SDI_pin(pin) or SPI3_SDI_pin(pin)

Syntax	SPI1_SDI_pin(pin); or SPI2_SDI_pin(pin); or SPI3_SDI_pin(pin);																																																																							
Arguments	pin																																																																							
	Specifies the pin to be set for SDI for SPI1, SPI2 or SPI3 ports.																																																																							
	pin	<table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>H1 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>1</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>3</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>5</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>7</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>29</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>27</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>25</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>23</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>21</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>19</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>8</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>6</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>28</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>30</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>24</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>26</td><td>No</td></tr> </tbody> </table>			4D Pin Name (Predefined)	DIABLO16 Pin Number	H1 Pin Number	Availability	PA0	61	1	Yes	PA1	62	3	Yes	PA2	63	5	Yes	PA3	64	7	Yes	PA4	46	29	Yes	PA5	49	27	Yes	PA6	50	25	Yes	PA7	51	23	Yes	PA8	52	21	Yes	PA9	53	19	Yes	PA10	43	8	Yes	PA11	44	6	Yes	PA12	31	28	Yes	PA13	32	30	Yes	PA14	37	24	No	PA15	36	26	No
4D Pin Name (Predefined)		DIABLO16 Pin Number	H1 Pin Number	Availability																																																																				
PA0		61	1	Yes																																																																				
PA1		62	3	Yes																																																																				
PA2		63	5	Yes																																																																				
PA3		64	7	Yes																																																																				
PA4		46	29	Yes																																																																				
PA5		49	27	Yes																																																																				
PA6		50	25	Yes																																																																				
PA7		51	23	Yes																																																																				
PA8		52	21	Yes																																																																				
PA9		53	19	Yes																																																																				
PA10		43	8	Yes																																																																				
PA11		44	6	Yes																																																																				
PA12		31	28	Yes																																																																				
PA13		32	30	Yes																																																																				
PA14	37	24	No																																																																					
PA15	36	26	No																																																																					
Returns	status																																																																							
	status	Returns TRUE if finction succeeded (usually ignored)																																																																						
Description	<p>Selects the hardware pin for SPI Receive line. SPI1, SPI2 or SPI3's SDI pin could be assigned to the available pins. Note that only a single pin should be mapped to spi SDI. If the pin argument is 0 the function has no effect. The pin is automatically set to an output.</p> <p>Note: If the spi SDI pin is set to same pin as spi SDO pin (eg for a loopback check) it is necessary to configure the SDI pin first, SPI2_SDI_pin(PA3); // configure SPI2 SDI to PA3 (this disconnects anything else) SPI2_SDO_pin(PA3)); // configure SPI2 SDO to PA3</p>																																																																							

2.10.10. SPI1_SDO_pin(pin) or SPI2_SDO_pin(pin) or SPI3_SDO_pin(pin)

Syntax	SPI1_SDO_pin(pin); or SPI2_SDO_pin(pin); or SPI3_SDO_pin(pin);				
Arguments	pin				
	pin	Specifies the pin to be set for SDO for SPI1, SPI2 or SPI3 ports.			
		4D Pin Name (Predefined)	DIABLO16 Pin Number	H1 Pin Number	Availability
		PA0	61	1	No
		PA1	62	3	Yes
		PA2	63	5	No
		PA3	64	7	Yes
		PA4	46	29	Yes
		PA5	49	27	Yes
		PA6	50	25	Yes
		PA7	51	23	Yes
		PA8	52	21	Yes
		PA9	53	19	Yes
		PA10	43	8	No
		PA11	44	6	No
		PA12	31	28	Yes
		PA13	32	30	Yes
	PA14	37	24	No	
	PA15	36	26	No	
Returns	status				
	status	Returns TRUE if finction succeeded (usually ignored)			
Description	Selects the hardware pin for SPI Transmit line. SPI1, SPI2 or SPI3's SDO pin could be assigned to the available pins. Note that only a single pin should be mapped to spi SDO. If the pin argument is 0 the previously selected spi SDO pin is disconnected. The pin is automatically set to an output.				

2.10.11. spi_ReadBlock() or spi1_ReadBlock() or spi2_ReadBlock() or spi3_ReadBlock()

Syntax	spi_ReadBlock("buf", "bufsize") or spi1_ReadBlock("buf", "bufsize") or spi2_ReadBlock("buf", "bufsize") or spi3_ReadBlock("buf", "bufsize")	
Arguments	buf, bufsize	
	Buf	String Pointer address of buffer to receive the data.
	Bufsize	The number of characters to receive into the buffer.
Returns	Nothing	
Description	Bufsize bytes are read from the SPI port to the string pointer "buf". This function gives much better performance than reading individual bytes at a time. Once the data has been read into a buffer it also makes it easy to perform CRC calculations on the data. The SPI port must be initialised in 8 bit mode.	
Example	<pre>#platform "uLCD-32WDT" func main() var st[20] ; // setup of spi3 pins and spi3 init goes here. Spi3_ReadBlock(str_Ptr(st), 8) ; // read 8 bytes from spi print(">", [STR] st, "<") ; // assumes bytes read are terminated repeat // maybe replace forever // this as well endfunc</pre>	

2.10.12. spi_WriteBlock() or spi1_WriteBlock() or spi2_WriteBlock() or spi3_WriteBlock()

Syntax	spi_WriteBlock("buf", "bufsize") or spi1_WriteBlock("buf", "bufsize") or spi2_WriteBlock("buf", "bufsize") or spi3_WriteBlock("buf", "bufsize")	
Arguments	buf, bufsize	
	Buf	String Pointer address of buffer to send the data from.
	Bufsize	The number of characters to send.
Returns	Nothing	
Description	Bufsize bytes are written to the SPI port from the string pointer "buf". This function gives much better performance than writing individual bytes at a time. Once the data has been read into a buffer it also makes it easy to perform CRC calculations on the data. The SPI port must be initialised in 8 bit mode.	
Example	<pre>#platform "uLCD-32WDT" func main() var st[20] ; to(st) ; print("Hello there!") ; // setup of spi3 pins and spi3 init goes here. spi3_WriteBlock(str_Ptr(st), 12) ; repeat // maybe replace forever // this as well endfunc</pre>	

2.11. Serial (UART) Communications Functions

Summary of Functions in this section:

- COM1_RX_pin(pin) **or** COM2_RX_pin(pin) **or** COM3_RX_pin(pin)
- COM1_TX_pin(pin) **or** COM2_TX_pin(pin) **or** COM3_TX_pin(pin)
- setbaud(rate)
- com_SetBaud(comport, baudrate/10)
- serin() **or** serin1() **or** serin2() **or** serin3()
- serout(char) **or** serout1(char) **or** serout2(char) **or** serout3(char)
- com_Init(buffer, bufsize, qualifier) **or** com_Init1(buffer, bufsize, qualifier) **or**

- com_Init2(buffer, bufsize, qualifier) **or** com_Init3(buffer, bufsize, qualifier)
- com_Reset() **or** com1_Reset() **or** com2_Reset() **or** com3_Reset()
- com_Count() **or** com1_Count() **or** com2_Count() **or** com3_Count()
- com_Full() **or** com1_Full() **or** com2_Full() **or** com3_Full()
- com_Error() **or** com1_Error() **or** com2_Error() **or** com3_Error()
- com_Sync() **or** com1_Sync() **or** com2_Sync() **or** com3_Sync()
- com_TXbuffer(buf, bufsize, pin) **or** com1_TXbuffer(buf, bufsize, pin) **or** com2_TXbuffer(buf, bufsize, pin) **or** com3_TXbuffer(buf, bufsize, pin)
- com_TXbufferHold(state) **or** com1_TXbufferHold(state) **or** com2_TXbufferHold(state) **or** com3_TXbufferHold(state)
- com_TXcount() **or** com1_TXcount() **or** com2_TXcount() **or** com3_TXcount()
- com_TXemptyEvent(function) **or** com1_TXemptyEvent(function) **or** com2_TXemptyEvent(function) **or** com3_TXemptyEvent(function)
- com_Mode()
- com_RXblock() **or** com1_RXblock() **or** com2_RXblock() **or** com3_RXblock()
- com_TXblock() **or** com1_TXblock() **or** com2_TXblock() **or** com3_TXblock()

2.11.1. COM1_RX_pin(pin) or COM2_RX_pin(pin) or COM3_RX_pin(pin)

Syntax	<code>COM1_RX_pin(pin);</code> or <code>COM2_RX_pin(pin);</code> or <code>COM3_RX_pin(pin);</code>																																																				
Arguments	pin																																																				
	pin	Specifies the GPIO pin to use for the com ports receive line																																																			
	The arguments can be a variable, array element, expression or constant																																																				
Returns	Status																																																				
	Status	Returns True if the function succeeded, usually ignored																																																			
Description	<p>Use this function to specify which GPIO is going to be assigned to the relative com ports receive line.</p> <p>Note that only a single pin can be mapped to any given com ports RX.</p> <p>If the pin argument is 0 the function has no effect.</p> <p>The pin is automatically set to an input. If the COMx RX pin is set to same pin as COMx TX pin (eg for a loopback check) it is necessary to configure the input pin first,</p> <p>For Example:</p> <pre>COM1_RX_pin(PA7); // config COM1 RX to PA7 (disconnects anything else) COM1_TX_pin(PA7)); // configure COM1 TX to PA7</pre> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>Yes</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>Yes</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>Yes</td></tr> <tr><td>PA11</td><td>44</td><td>Yes</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	Yes	PA1	62	Yes	PA2	63	Yes	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	Yes	PA11	44	Yes	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	Yes																																																			
PA1	62	Yes																																																			
PA2	63	Yes																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	Yes																																																			
PA11	44	Yes																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	<code>COM1_RX_pin(PA7); // config COM1 RX to PA7</code>																																																				

2.11.2. COM1_TX_pin(pin) or COM2_TX_pin(pin) or COM3_TX_pin(pin)

Syntax	COM1_TX_pin(pin); or COM2_TX_pin(pin); or COM3_TX_pin(pin);																																																				
Arguments	pin																																																				
	pin	Specifies the GPIO pin to use for the com ports transmit line																																																			
	The arguments can be a variable, array element, expression or constant																																																				
Returns	Status																																																				
	Status	Returns True if the function succeeded, usually ignored																																																			
Description	<p>Use this function to specify which GPIO is going to be assigned to the relative com ports transmit line.</p> <p>Note that only a single pin can be mapped to any given com ports TX.</p> <p>If the pin argument is 0, COMx TX is disconnected from all pins.</p> <p>The pin is automatically set to an output.</p> <table border="1"> <thead> <tr> <th>4D Pin Name (Predefined)</th> <th>DIABLO16 Pin Number</th> <th>Availability</th> </tr> </thead> <tbody> <tr><td>PA0</td><td>61</td><td>No</td></tr> <tr><td>PA1</td><td>62</td><td>Yes</td></tr> <tr><td>PA2</td><td>63</td><td>No</td></tr> <tr><td>PA3</td><td>64</td><td>Yes</td></tr> <tr><td>PA4</td><td>46</td><td>Yes</td></tr> <tr><td>PA5</td><td>49</td><td>Yes</td></tr> <tr><td>PA6</td><td>50</td><td>Yes</td></tr> <tr><td>PA7</td><td>51</td><td>Yes</td></tr> <tr><td>PA8</td><td>52</td><td>Yes</td></tr> <tr><td>PA9</td><td>53</td><td>Yes</td></tr> <tr><td>PA10</td><td>43</td><td>No</td></tr> <tr><td>PA11</td><td>44</td><td>No</td></tr> <tr><td>PA12</td><td>31</td><td>Yes</td></tr> <tr><td>PA13</td><td>32</td><td>Yes</td></tr> <tr><td>PA14</td><td>37</td><td>No</td></tr> <tr><td>PA15</td><td>36</td><td>No</td></tr> </tbody> </table>		4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability	PA0	61	No	PA1	62	Yes	PA2	63	No	PA3	64	Yes	PA4	46	Yes	PA5	49	Yes	PA6	50	Yes	PA7	51	Yes	PA8	52	Yes	PA9	53	Yes	PA10	43	No	PA11	44	No	PA12	31	Yes	PA13	32	Yes	PA14	37	No	PA15	36	No
4D Pin Name (Predefined)	DIABLO16 Pin Number	Availability																																																			
PA0	61	No																																																			
PA1	62	Yes																																																			
PA2	63	No																																																			
PA3	64	Yes																																																			
PA4	46	Yes																																																			
PA5	49	Yes																																																			
PA6	50	Yes																																																			
PA7	51	Yes																																																			
PA8	52	Yes																																																			
PA9	53	Yes																																																			
PA10	43	No																																																			
PA11	44	No																																																			
PA12	31	Yes																																																			
PA13	32	Yes																																																			
PA14	37	No																																																			
PA15	36	No																																																			
Example	COM1_TX_pin(PA7); // config COM1 RX to PA7																																																				

2.11.3. setbaud(baudnum)

Syntax	setbaud(baudnum);																																																																																						
Arguments	baudnum																																																																																						
	baudnum	Specifies the baud rate of COM0 using the baud number or pre-defined constant																																																																																					
		<table border="1"> <thead> <tr> <th>Baud number</th> <th>Pre-defined Constant</th> <th>Baud Rate Error (%)</th> <th>Actual Baud Rate</th> </tr> </thead> <tbody> <tr><td>0</td><td>BAUD_110</td><td>0.00%</td><td>110</td></tr> <tr><td>1</td><td>BAUD_300</td><td>0.00%</td><td>300</td></tr> <tr><td>2</td><td>BAUD_600</td><td>0.00%</td><td>600</td></tr> <tr><td>3</td><td>BAUD_1200</td><td>0.00%</td><td>1200</td></tr> <tr><td>4</td><td>BAUD_2400</td><td>0.04%</td><td>2401</td></tr> <tr><td>5</td><td>BAUD_4800</td><td>0.04%</td><td>4802</td></tr> <tr><td>6</td><td>BAUD_9600</td><td>0.16%</td><td>9615</td></tr> <tr><td>7</td><td>BAUD_14400</td><td>0.27%</td><td>14439</td></tr> <tr><td>8</td><td>BAUD_19200</td><td>0.38%</td><td>19273</td></tr> <tr><td>9</td><td>BAUD_31250 or MIDI</td><td>0.00%</td><td>31250</td></tr> <tr><td>10</td><td>BAUD_38400</td><td>0.83%</td><td>38717</td></tr> <tr><td>11</td><td>BAUD_56000</td><td>0.16%</td><td>56090</td></tr> <tr><td>12</td><td>BAUD_57600</td><td>1.27%</td><td>58333</td></tr> <tr><td>13</td><td>BAUD_115200</td><td>2.64%</td><td>118243</td></tr> <tr><td>14</td><td>BAUD_128000</td><td>0.53%</td><td>128676</td></tr> <tr><td>15</td><td>BAUD_256000</td><td>0.53%</td><td>257353</td></tr> <tr><td>16</td><td>BAUD_300000</td><td>4.17%</td><td>312500</td></tr> <tr><td>17</td><td>BAUD_375000</td><td>6.06%</td><td>397727</td></tr> <tr><td>18</td><td>BAUD_500000</td><td>9.38%</td><td>546875</td></tr> <tr><td>19</td><td>BAUD_600000</td><td>4.17%</td><td>625000</td></tr> </tbody> </table>	Baud number	Pre-defined Constant	Baud Rate Error (%)	Actual Baud Rate	0	BAUD_110	0.00%	110	1	BAUD_300	0.00%	300	2	BAUD_600	0.00%	600	3	BAUD_1200	0.00%	1200	4	BAUD_2400	0.04%	2401	5	BAUD_4800	0.04%	4802	6	BAUD_9600	0.16%	9615	7	BAUD_14400	0.27%	14439	8	BAUD_19200	0.38%	19273	9	BAUD_31250 or MIDI	0.00%	31250	10	BAUD_38400	0.83%	38717	11	BAUD_56000	0.16%	56090	12	BAUD_57600	1.27%	58333	13	BAUD_115200	2.64%	118243	14	BAUD_128000	0.53%	128676	15	BAUD_256000	0.53%	257353	16	BAUD_300000	4.17%	312500	17	BAUD_375000	6.06%	397727	18	BAUD_500000	9.38%	546875	19	BAUD_600000	4.17%	625000	
Baud number	Pre-defined Constant	Baud Rate Error (%)	Actual Baud Rate																																																																																				
0	BAUD_110	0.00%	110																																																																																				
1	BAUD_300	0.00%	300																																																																																				
2	BAUD_600	0.00%	600																																																																																				
3	BAUD_1200	0.00%	1200																																																																																				
4	BAUD_2400	0.04%	2401																																																																																				
5	BAUD_4800	0.04%	4802																																																																																				
6	BAUD_9600	0.16%	9615																																																																																				
7	BAUD_14400	0.27%	14439																																																																																				
8	BAUD_19200	0.38%	19273																																																																																				
9	BAUD_31250 or MIDI	0.00%	31250																																																																																				
10	BAUD_38400	0.83%	38717																																																																																				
11	BAUD_56000	0.16%	56090																																																																																				
12	BAUD_57600	1.27%	58333																																																																																				
13	BAUD_115200	2.64%	118243																																																																																				
14	BAUD_128000	0.53%	128676																																																																																				
15	BAUD_256000	0.53%	257353																																																																																				
16	BAUD_300000	4.17%	312500																																																																																				
17	BAUD_375000	6.06%	397727																																																																																				
18	BAUD_500000	9.38%	546875																																																																																				
19	BAUD_600000	4.17%	625000																																																																																				
	The arguments can be a variable, array element, expression or constant																																																																																						
Returns	nothing																																																																																						
Description	<p>Use this function to set the required baud rate. The default Baud Rate for COM0 is 115,200 bits per second or 115,200 baud.</p> <p>If a value other than 0-19 is used, a run time error (error 25)</p>																																																																																						

2.11.4. com_SetBaud(comport, baudrate/10)

Syntax	<code>com_SetBaud("comport", "baudrate/10");</code>	
Arguments	comport, baudrate/10	
	comport	Specifies the Com port, COM0: COM1: COM2: COM3:
	baudrate/10	Specifies the baud rate.
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Returns True if BAUD rate was acceptable.
Description	<p>Use this function to set the required baud rate for the required Com port. Sets to any viable baud rate from 160 to 655350.</p> <p>Note: The default Baud Rate for COM0 is 115,200 bits per second or 115,200 baud. The default Baud Rate for COM1, COM2 and COM3 is 9600 bits per second or 9600 baud.</p> <p>Note: As of the v1.1 PmmC several 'low' values have special meanings</p> <ul style="list-style-type: none"> 1 : 2187500 baud 2 : 1458333 baud 3 : 1093750 baud 4 : 875000 baud 5 : 729167 baud 	
Example	<pre>stat := com_SetBaud(COM2, 960) // To set Com2 to 9600 BAUD rate. if (stat) Print("Com2 set to 9600 BAUD"); endif</pre>	

2.11.5. `serin()` or `serin1()` or `serin2()` or `serin3()`

Syntax	<code>serin();</code> or <code>serin1();</code> or <code>serin2();</code> or <code>serin3();</code>	
Arguments	none	
Returns	char	
	char	Returns: -1 if no character is available Returns: -2 if a framing error or over-run has occurred (auto cleared) Returns: positive value 0 to 255 for a valid character received
Description	<p><code>serin()</code>: Receives a character from the Serial Port COM0. <code>serin1()</code>: Receives a character from the Serial Port COM1. <code>serin2()</code>: Receives a character from the Serial Port COM2. <code>serin3()</code>: Receives a character from the Serial Port COM3.</p> <p><code>serin</code> may be buffered (refer to <code>com_Init(..)</code> functions)</p> <p>The transmission format is: No Parity, 1 Stop Bit, 8 Data Bits (N,8,1).</p> <p>Note: COM0 pins cannot be mapped, and are fixed as pins 42(Rx0) and 33(Tx0) on the DIABLO cihp. Rx and Tx of COM1, COM2 or COM3 should be defined before using <code>serin1()</code>, <code>serin2()</code> or <code>serin3()</code>.</p>	
Example	<pre>var char; char := serin(); // test the com0 port if (char >= 0) // if a valid character is received process(char); // process the character endif</pre>	

2.11.6. serout(char) or serout1(char) or serout2(char) or serout3(char)

Syntax	<code>serout(char);</code> or <code>serout1(char);</code> or <code>serout2(char);</code> or <code>serout3(char);</code>	
Arguments	<code>char</code>	
	<code>char</code>	Specifies the data byte to be sent to the serial port.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p><code>serout()</code>: Transmits a single byte to the Serial Port COM0. <code>serout1()</code>: Transmits a single byte to the Serial Port COM1. <code>serout2()</code>: Transmits a single byte to the Serial Port COM2. <code>serout3()</code>: Transmits a single byte to the Serial Port COM3.</p> <p>The transmission format is: No Parity, 1 Stop Bit, 8 Data Bits (N,8,1).</p> <p>Note: COM0 pins cannot be mapped, and are fixed as pins 42(Rx0) and 33(Tx0) on the DIABLO cihp. Rx and Tx of COM1, COM2 or COM3 should be defined before using <code>serout1()</code>, <code>serout2()</code> or <code>serout3()</code>.</p>	
Example	<code>serout('\n');</code> <code>\\Send a linefeed to COM0.</code>	

2.11.7. `com_Init(buffer, bufsize, qualifier)` or `com1_Init(buffer, bufsize, qualifier)` or `com2_Init(buffer, bufsize, qualifier)` or `com3_Init(buffer, bufsize, qualifier)`

Syntax	<code>com_Init(buffer, bufsize, qualifier);</code> or <code>com1_Init(buffer, bufsize, qualifier);</code> or <code>com2_Init(buffer, bufsize, qualifier);</code> or <code>com3_Init(buffer, bufsize, qualifier);</code>	
Arguments	buffer, bufsize, qualifier	
	buffer	Specifies the address of a buffer used for the background buffering service.
	bufsize	Specifies the byte size of the user array provided for the buffer (each array element holds 2 bytes). If the buffer size is zero, a buffer of 128 words (256 bytes) should be provided for automatic packet length mode (see below).
	qualifier	Specifies the qualifying character that must be received to initiate serial data reception and buffer write. A zero (0x00) indicates no qualifier to be used.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	<p>This is the initialisation function for the serial communications buffered service. Once initialised, the service runs in the background capturing and buffering serial data without the user application having to constantly poll the serial port. This frees up the application to service other tasks.</p> <p><u>MODES OF OPERATION</u></p> <ul style="list-style-type: none"> <u>No qualifier – simple ring buffer (aka circular queue)</u> <p>If the <i>qualifier</i> is set to zero, the <i>buffer</i> is continually active as a simple circular queue. Characters when received from the host are placed in the circular queue (at the 'head' of the queue) Bytes may be removed from the circular queue (from the 'tail' of the queue) using the <code>serin()</code> function. If the tail is the same position as the head, there are no bytes in the queue, therefore <code>serin()</code> will return -1, meaning no character is available, also, the <code>com_Count()</code> function can be read at any time to determine the number of characters that are waiting between the tail and head of the queue. If the queue is not read frequently by the application, and characters are still being sent by the host, the head will eventually catch up with the tail setting the internal <code>COM_FULL</code> flag (which can be read with the <code>com_Full()</code> function) . Any further characters from the host are now discarded, however, all the characters that were buffered up to this point are readable. This is a good way of reading a fixed size packet and not necessarily considered to be an error condition. If no characters are removed from the buffer until the <code>COM_FULL</code> flag (which can be read with the <code>com_Full()</code> function) becomes set, it is guaranteed that the bytes will be ordered in the <i>buffer</i> from the start position, therefore, the <i>buffer</i> can be treated as an array and can be read directly without using <code>serin()</code> at all. In the latter case, the correct action is to process the data from the buffer, re-initialise the buffer with the <code>com_Init(..)</code> function, or reset the buffered serial service by issuing the <code>com_Reset()</code> function (which will return serial reception to polled mode) , and send an acknowledgement to the host (traditionally a ACK or 6) to indicate that the application is ready to receive more data and the previous 'packet' has been dealt with, or conversely, the application may send a negative acknowledgement to indicate that some sort of error occurred, or the action could not be completed (traditionally a NAK or 16) .</p>	

If any low level errors occur during the buffering service (such as framing or over-run) the internal COM_ERROR flag will be set (which can be read with the **com_Error()** function). Note that the COM_FULL flag will remain latched to indicate that the buffer did become full, and is not reset (even if all the characters are read) until the **com_Init(..)** or **com_Reset()** function is issued.

- **Using a qualifier**

If a **qualifier** character is specified, after the buffer is initialised with **com_Init(..)**, the service will ignore all characters until the **qualifier** is received and only then initiate the buffer write sequence with incoming data. After that point, the behaviour is the same as above for the 'non qualified' mode.

com_Init(buffer, bufsize, qualifier): Initialize a serial capture buffer for COM0.

com1_Init(buffer, bufsize, qualifier): Initialize a serial capture buffer for COM1.

com2_Init(buffer, bufsize, qualifier): Initialize a serial capture buffer for COM2.

com3_Init(buffer, bufsize, qualifier): Initialize a serial capture buffer for COM3.

Example

```
com_Init(combuf, 20, 0 );
//set up a comms ring buffer for COM0, 20 characters before overflow
```

2.11.8. com_Reset() or com1_Reset() or com2_Reset() or com3_Reset()

Syntax	<code>com_Reset();</code> or <code>com1_Reset();</code> or <code>com2_Reset();</code> or <code>com3_Reset();</code>
Arguments	none
Returns	nothing
Description	Resets the serial communications buffered service and returns it to the default polled mode. <code>com_Reset()</code> Reset COM0 <code>com1_Reset()</code> Reset COM1 <code>com2_Reset()</code> Reset COM2 <code>com3_Reset()</code> Reset COM3
Example	<code>com_Reset(); // reset COM0 to polled mode</code>

2.11.9. `com_Count()` or `com1_Count()` or `com2_Count()` or `com3_Count()`

Syntax	<code>com_Count();</code> or <code>com1_Count();</code> or <code>com2_Count();</code> or <code>com3_Count();</code>	
Arguments	none	
Returns	count	
	count	Current count of characters in the communications buffer.
Description	Can be read at any time (when in buffered communications is active) to determine the number of characters that are waiting in the buffer. <code>com_Count();</code> Charcters countr in COM0 <code>com1_Count();</code> Charcters countr in COM1 <code>com2_Count();</code> Charcters countr in COM2 <code>com3_Count();</code> Charcters countr in COM3	
Example	<code>n := com_Count(); // get the number of chars available in the buffer</code>	

2.11.10. `com_Full()` or `com1_Full()` or `com2_Full()` or `com3_Full()`

Syntax	<code>com_Full();</code> or <code>com1_Full();</code> or <code>com2_Full();</code> or <code>com3_Full();</code>	
Arguments	none	
Returns	status	
	status	Returns 1 if buffer or queue has become full, or is overflowed, else returns 0 .
Description	If the queue is not read frequently by the application, and characters are still being sent by the host, the head will eventually catch up with the tail setting the COM_FULL flag which is read with this function. If this flag is set, any further characters from the host are discarded, however, all the characters that were buffered up to this point are readable.	
Example	<pre>if(com_Full() & (com_Count() == 0)) com_Init(mybuf, 30, 0); // buffer full, recovery endif</pre>	

2.11.11. com_Error() or com1_Error() or com2_Error() or com3_Error()

Syntax	<code>com_Error();</code> or <code>com1_Error();</code> or <code>com2_Error();</code> or <code>com3_Error();</code>	
Arguments	none	
Returns	status	
	status	Returns 1 if any low level communications error occurred, else returns 0 .
Description	If any low level errors occur during the buffering service (such as framing or over-run) the internal COM_ERROR flag will be set which can be read with this function.	
Example	<pre>if(com_Error()) // if there were low level comms errors, resetMySystem(); // take corrective action endif</pre>	

2.11.12. `com_Sync()` or `com1_Sync()` or `com2_Sync()` or `com3_Sync()`

Syntax	<code>com_Sync();</code> or <code>com1_Sync();</code> or <code>com2_Sync();</code> or <code>com3_Sync();</code>	
Arguments	none	
Returns	status	
	status	Returns 1 if the qualifier character has been received, else returns 0 .
Description	If a <i>qualifier</i> character is specified when using buffered communications, after the buffer is initialized with <code>com_Init(..)</code> , <code>com1_Init(..)</code> , <code>com2_Init(..)</code> , or <code>com3_Init(..)</code> the service will ignore all characters until the <i>qualifier</i> is received and only then initiate the buffer write sequence with incoming data. <code>com_Sync()</code> , <code>com1_Sync()</code> , <code>com2_Sync()</code> , <code>com3_Sync()</code> is called to determine if the qualifier character has been received yet.	
Example	<code>stat := com_Sync(); // See if the qualifier is received at COM0</code>	

2.11.13. `com_TXbuffer(buf, bufsize, pin)` or `com1_TXbuffer(buf, bufsize, pin)` or `com2_TXbuffer(buf, bufsize, pin)` or `com3_TXbuffer(buf, bufsize, pin)`

Syntax	<code>com_TXbuffer(buf, bufsize, pin);</code> or <code>com1_TXbuffer(buf, bufsize, pin);</code> or <code>com2_TXbuffer(buf, bufsize, pin);</code> or <code>com3_TXbuffer(buf, bufsize, pin);</code>	
Arguments	<code>buf, bufsize, pin</code>	
	buf	Specifies the address of a buffer used for the buffering service.
	bufsize	Specifies the byte size of the user array provided for the buffer (each array element holds 2 bytes).
	pin	Specifies the turnaround pin. If not required, just set "pin" to zero.
	The arguments can be a variable, array element, expression or constant	
Returns	None	
Description	<p>Initialise a serial buffer for the COM0, COM1, COM2 or COM3 output. The program must declare a var array as a circular buffer. When a TX buffer is declared for comms, the transmission of characters becomes non-blocking. If the buffer has insufficient space to accept the next character from a <code>serout(..)</code>, <code>serout1(..)</code>, <code>serout2(..)</code> or <code>serout3(..)</code> function, the excess characters will be ignored, and the <code>com_Full()</code>, <code>com1_Full()</code>, <code>com2_Full()</code> or <code>com3_Full()</code> error will be asserted. If the TX buffer is no longer required, just set the buffer pointer to zero, the size in this case doesnt matter and is ignored. The function can be resized or reallocated to another buffer at any time. The buffer is flushed before any changes are made.</p> <p>"pin" designates an IO pin to control a bi-directional control device for half duplex mode. "pin" will go HI at the start of a transmission, and will return low after the final byte is transmitted.</p>	
Example	<pre>com_TXbuffer(mybuf, 1024, PA1); // set the TX buffer of COM0 com_TXbuffer(0, 0, 0); // revert COM0 to non buffered service</pre>	

2.11.14. `com_TXbufferHold(state)` or `com1_TXbufferHold(state)` or `com2_TXbufferHold(state)` or `com3_TXbufferHold(state)`

Syntax	<code>com_TXbufferHold(state);</code> or <code>com1_TXbufferHold(state);</code> or <code>com2_TXbufferHold(state);</code> or <code>com3_TXbufferHold(state);</code>	
Arguments	state	
	state	Specifies the state of the buffer used for the buffering service.
	The arguments can be a variable, array element, expression or constant	
Returns	count	
	count	Returns -1 if function is called illegally when TX comms is not buffered. Returns buffer count when called with argument of 1, for example <code>com_TXbufferHold(ON)</code> , <code>com1_TXbufferHold(ON)</code> , <code>com2_TXbufferHold(ON)</code> or <code>com3_TXbufferHold(ON)</code> Returns 0 when argument is zero, eg <code>com_TXbufferHold(OFF)</code> , <code>com1_TXbufferHold(OFF)</code> , <code>com2_TXbufferHold(OFF)</code> , <code>com3_TXbufferHold(OFF)</code>
Description	<p>This function is used in conjunction with <code>com_TXbuffer(...)</code>, <code>com1_TXbuffer(...)</code>, <code>com2_TXbuffer(...)</code>, <code>com3_TXbuffer(...)</code>, .</p> <p>It is often necessary to hold off sending serial characters until a complete frame or packet has been built in the output buffer. <code>com_TXbufferHold(ON)</code>, <code>com1_TXbufferHold(ON)</code>, <code>com2_TXbufferHold(ON)</code>, <code>com3_TXbufferHold(ON)</code> is used for this, to stop the buffer being sent while it is being loaded. Normally, when using buffered comms, the transmit process will begin immediately. This is fine unless you are trying to assemble a packet.</p> <p>To build a packet and send it later, issue a <code>com_TXbufferHold(ON)</code>, <code>com1_TXbufferHold(ON)</code>, <code>com2_TXbufferHold(ON)</code>, <code>com3_TXbufferHold(ON)</code> build the packet, when packet is ready, issuing <code>com_TXbufferHold(OFF)</code>, <code>com1_TXbufferHold(OFF)</code>, <code>com2_TXbufferHold(OFF)</code>, <code>com3_TXbufferHold(OFF)</code> will release the buffer to the com port.</p> <p>Also, if using <code>com_TXemptyEvent</code>, <code>com1_TXemptyEvent</code>, <code>com2_TXemptyEvent</code>, <code>com3_TXemptyEvent</code>, erroneous empty events will occur as the transmit buffer is constantly trying to empty while you are busy trying to fill it.</p> <p>Also refer to the pin control for <code>com_TXbuffer(..)</code>, <code>com1_TXbuffer(..)</code>, <code>com2_TXbuffer(..)</code>, <code>com3_TXbuffer(..)</code> function.</p> <p>Note: If you fill the buffer whilst it is held comms error 4 will be set and the data written will be lost.</p>	
Example	Refer to the <code>com_TXemptyEvent(functionAddress)</code> example.	

2.11.15. `com_TXcount()` or `com1_TXcount()` or `com2_TXcount()` or `com3_TXcount()`

Syntax	<code>com_TXcount();</code> or <code>com1_TXcount();</code> or <code>com2_TXcount();</code> or <code>com3_TXcount();</code>	
Arguments	None	
Returns	count	
	count	Returns count of characters
Description	Return count of characters remaining in COM0 , COM1 or COM2 or COM3 transmit buffer that was previously allocated with <code>com_TXbuffer(..)</code> , <code>com1_TXbuffer(..)</code> , <code>com2_TXbuffer(..)</code> , <code>com3_TXbuffer(..)</code> .	
Example	<code>arg := com1_TXCount(); //return count of characters in COM1 TX buffer</code>	

2.11.16. com_TXemptyEvent(function) or com1_TXemptyEvent(function) or com2_TXemptyEvent(function) or com3_TXemptyEvent(function)

Syntax	com_TXemptyEvent(functionAddress); or com1_TXemptyEvent(functionAddress); or com2_TXemptyEvent(functionAddress); or com3_TXemptyEvent(functionAddress);	
Arguments	functionAddress	
	functionAddress	Address of the event Function to be queued when COM0, COM1, COM2 or COM3 TX buffer empty
Returns	Address	
	Address	Returns any previous event function address or zero if there was no previous function.
Description	<p>If a comms TX buffer that was previously allocated with com_TXbuffer(...), com1_TXbuffer(...), com2_TXbuffer(...) or com3_TXbuffer(...) this function can be used to set up a function to be called when the COM0, COM1, COM2 or COM3 TX buffer is empty.</p> <p>This is useful for either reloading the TX buffer, setting or clearing a pin to change the direction of eg a RS485 line driver, or any other form of traffic control. The event function must not have any parameters. To disable the event, simply call com_TXemptyEvent(0), com1_TXemptyEvent(0), com2_TXemptyEvent(0) or com3_TXemptyEvent(0).</p> <p>com_TXbuffer(...), com1_TXbuffer(...), com2_TXbuffer(...) or com3_TXbuffer(...) also resets any active event.</p>	
Example	<pre>#platform "uLCD-32PT_GFX2" /***** * Description: buffered TX service * Use Workshop terminal at 9600 baud to see result * Example of Buffered TX service vs Non buffered * Also explains the use of COMMS events * * NB Program must be written to flash so * the Workshop Terminal can be used. * *****/ var combuf[220]; // buffer for up to 440 bytes // run a timer event while we are doing comms func T7Service() var private colour := 0xF800; colour ^= 0xF800; gfx_RectangleFilled(50,200,80,220,colour); sys_SetTimer(TIMER7, 200); endfunc // event to capture the buffer empty event func bufEmpty() com_TXbuffer(0, 0, IO1_PIN); // done with the buffer, release it print("\n\nHELLO WORLD, I'M EMPTY ",com_TXcount(),"\n"); endfunc</pre>	


```

func main()
  var n, r, D, fh;

  sys_SetTimerEvent(TIMER7,T7Service);      // run a timer event
  sys_SetTimer(TIMER7, 150);
  com_TXemptyEvent(bufEmpty); // set to capture  buffer empty event

  setbaud(BAUD_9600);

  txt_Set(TEXT_OPACITY, OPAQUE);

repeat
  gfx_Cls();

  txt_MoveCursor(3,1);      // reset cursor to line 3, column 2
  print("Send 440 chars non-buffered\n");
  pokeW(SYSTEM_TIMER_LO, 0); // reset timer

  // note that 440 chars at 9600 baud takes approx 453msec
  for(n:=0; n<10; n++)
    to(COM0); putstr("The quick brown fox jumps over the lazy
dog\n"); // 44 chars
  next

  print("took ",peekW(SYSTEM_TIMER_LO),"Msec\n\n");
  // time spent blocking is only approx 1msec

  com_TXbuffer(combuf, 440,IO1_PIN); // set up the TX buffer
  com_TXbufferHold(ON);              // hold the TX buffer til ready

  // note that here the time is only approx 1msec overhead due to
buffering.
  print("Send 440 chars buffered\n");
  pokeW(SYSTEM_TIMER_LO, 0);        // reset timer

  for(n:=0; n<10; n++)
    to(COM0); putstr("THE QUICK BROWN FOX JUMPS OVER THE LAZY
DOG\n"); // 44 chars
  next

  print("took ",peekW(SYSTEM_TIMER_LO),"Msec\n\n");
  // time spent blocking is only approx 1msec

  // demonstrate how to modify a prepared comms buffer that is
still being held
  to(combuf); print("MY CONTENTS HAVE BEEN CHANGED");
  to(combuf+50); print("*** AND CHANGED HERE TOO ***");
  combuf[218] := 'CA'; // the last 'DOG' changed here
  combuf[219] := 'T\n'; // the last 'DOG' changed here

  // now we are ready to send to buffer
  n := com_TXbufferHold(OFF); // release TX buffer
  print("TXBuffer is holding ", n, " chars\n");
  // show how many characters were in the buffer

  // watch the buffer empty
  repeat
    print("TX count = ", [DEC5ZB] n := com_TXcount(),"\r"); //
watch the count as the buffer empties
    until(!n);

    print("\n\nTX Empty");

  com_TXbuffer(0, 0, IO1_PIN); // done with the buffer, release it

```

```
        sys_SetTimer(TIMER0, 3000); // pause for 3 seconds, non blocking
        while(peekW(TMR0));

    forever // do it forever
    //com_TXbuffer(0, 0, 0); // if done with the pin, must release it

endfunc
```

2.11.17. com_Mode("databits", "parity", "Stopbits", "comport")

Syntax	<code>com_Mode("databits", "parity", "Stopbits", "comport");</code>	
Arguments	Databits, parity, Stopbits, comport.	
	Databits	Specifies the number of databits, 8 is the only currently valid value
	Parity	Specifies the parity bit. Valid values are N(one), E(ven) and O(dd).
	Stopbits	Specifies the number of stop bits. Valid values are 1 and 2.
	Comport	Specifies the Com port, COM0: COM1: COM2: COM3:
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Returns True if the parameters were acceptable.
Description	Use this function to set the required serial port parameters to other than 8N1	
Example	<pre> stat := com_Mode(8, 'E', 2, COM2) // To set Com2 to 8E2. if (stat) Print("Com2 set to 8E2"); endif </pre>	

2.11.18. com_RXblock() or com1_RXblock() or com2_RXblock() or com3_RXblock()

Syntax	<code>com_RXblock("buf", "bufsize") or com1_RXblock("buf", "bufsize") or com2_RXblock("buf", "bufsize") or com3_RXblock("buf", "bufsize")</code>	
Arguments	buf, bufsize	
	Buf	String Pointer address of buffer to receive the data.
	Bufsize	The number of characters to receive into the buffer.
Returns	Nothing	
Description	Bufsize bytes are received from the serial port to the string pointer "buf". If a receive buffer is active and bufsize characters are available this function will return almost immediately otherwise it will block until until the required bytes are received. This function is useful for protocols that require the reading of a fixed amount of data in one hit. Once the data has been read into a buffer it also makes it easy to perform CRC calculations on the data.	
Example	<pre>#platform "uLCD-32WDT" func main() var st[20] ; com_RXblock(str_Ptr(st), 8) ; str_PutByte(str_Ptr(st)+8, 0) ; // terminate print(">", [STR] st, "<") ; repeat // maybe replace forever // this as well endfunc</pre>	

2.11.19. com_TXblock() or com1_TXblock() or com2_TXblock() or com3_TXblock()

Syntax	com_TXblock("buf", "bufsize") or com1_TXblock("buf", "bufsize") or com2_TXblock("buf", "bufsize") or com3_TXblock("buf", "bufsize")	
Arguments	buf, bufsize	
	Buf	String Pointer address of buffer to send the data from.
	Bufsize	The number of characters to send.
Returns	Nothing	
Description	Bufsize bytes are transmitted to the serial port from the string pointer "buf". If a transmit buffer is active and space is available this function will return almost immediately otherwise it will block until until the required bytes are sent. This function is useful for protocols that require the reading of a fixed amount of data in one hit. Once the data has been read into a buffer it also makes it easy to perform CRC calculations on the data.	
Example	<pre>#platform "uLCD-32WDT" func main() var st[20] ; to(st) ; print("Hello there!") ; com_TXblock(str_Ptr(st), 12) ; com_TXblock("\nThis is a Test", 15) ; repeat // maybe replace forever // this as well endfunc</pre>	

2.12. I2C BUS Master Functions

Summary of Functions in this section:

- I2C1_Open(Speed, SCLpin, SDApin) **or** I2C2_Open(Speed, SCLpin, SDApin) **or** I2C3_Open(Speed, SCLpin, SDApin)
- I2C1_Close() **or** I2C2_Close() **or** I2C3_Close()
- I2C1_Start() **or** I2C2_Start() **or** I2C3_Start()
- I2C1_Stop() **or** I2C2_Stop() **or** I2C3_Stop()
- I2C1_Restart() **or** I2C2_Restart() **or** I2C3_Restart()
- I2C1_Read() **or** I2C2_Read() **or** I2C3_Read()
- I2C1_Write(byte) **or** I2C2_Write(byte) **or** I2C3_Write(byte)
- I2C1_Ack() **or** I2C2_Ack() **or** I2C3_Ack()
- I2C1_Nack() **or** I2C2_Nack() **or** I2C3_Nack()
- I2C1_AckStatus() **or** I2C2_AckStatus() **or** I2C3_AckStatus()
- I2C1_AckPoll(control) **or** I2C2_AckPoll(control) **or** I2C3_AckPoll(control)
- I2C1_Idle() **or** I2C2_Idle() **or** I2C3_Idle()
- I2C1_Gets(buffer, size) **or** I2C2_Gets(buffer, size) **or** I2C3_Gets(buffer, size)
- I2C1_Getn(buffer, size) **or** I2C2_Getn(buffer, size) **or** I2C3_Getn(buffer, size)
- I2C1_Puts(buffer) **or** I2C2_Puts(buffer) **or** I2C3_Puts(buffer)
- I2C1_Putn(buffer, count) **or** I2C2_Putn(buffer, count) **or** I2C3_Putn(buffer, count)


2.12.1. I2C1_Open(Speed, SCL, SDA) or I2C2_Open(Speed, SCL, SDA) or I2C3_Open(Speed, SCL, SDA)

Syntax	I2C1_Open(Speed, SCLpin, SDAPin); or I2C2_Open(Speed, SCLpin, SDAPin); or I2C3_Open(Speed, SCLpin, SDAPin);																	
Arguments	Speed																	
	Speed	Specifies the I ² C bus speed (See list in Description box)																
	SCLpin	Specifies the GPIO pin to use for the SCL signal																
	SDAPin	Specifies the GPIO pin to use for the SDA signal																
	The arguments can be a variable, array element, expression or constant																	
Returns	Status																	
	Status	1 if Successful 0 if Unsuccessful																
Description	<p>Calling this function configures the I²C module and initialises it to be ready for service. The I²C clock speed is specified by the Speed parameter. Multiple I²C Speed settings are available to suit various requirements.</p> <table border="1"> <thead> <tr> <th>Constant</th> <th>Speed</th> </tr> </thead> <tbody> <tr> <td>I2C_SLOW</td> <td>100KHz</td> </tr> <tr> <td>I2C_MED</td> <td>400KHz</td> </tr> <tr> <td>I2C_FAST</td> <td>1MHz</td> </tr> <tr> <td>I2C_10KHZ</td> <td>10KHz</td> </tr> <tr> <td>I2C_20KHZ</td> <td>20KHz</td> </tr> <tr> <td>I2C_50KHZ</td> <td>50KHz</td> </tr> <tr> <td>I2C_250KHZ</td> <td>250KHz</td> </tr> </tbody> </table> <p>Note: Normally the I2C pins are PA0 to PA13, use of these pins has a couple of limitations, a) There is no slew rate control at I2C_MED and b) I2C_FAST is not truly 1MHz. If either of these restrictions need to be addressed, a special case of SCLpin = PA14 and SDAPin = PA15 exists ONLY for speeds I2C_MED (which uses slew rate control) and I2C_FAST (which is truly 1MHz).</p>		Constant	Speed	I2C_SLOW	100KHz	I2C_MED	400KHz	I2C_FAST	1MHz	I2C_10KHZ	10KHz	I2C_20KHZ	20KHz	I2C_50KHZ	50KHz	I2C_250KHZ	250KHz
Constant	Speed																	
I2C_SLOW	100KHz																	
I2C_MED	400KHz																	
I2C_FAST	1MHz																	
I2C_10KHZ	10KHz																	
I2C_20KHZ	20KHz																	
I2C_50KHZ	50KHz																	
I2C_250KHZ	250KHz																	
Example	I2C1_Open(I2C_MED, PA2, PA3); // Open the I ² C port in 400KHz mode.																	

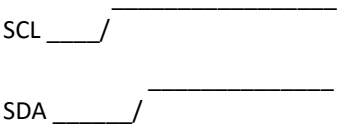
2.12.2. I2C1_Close() or I2C2_Close() or I2C3_Close()

Syntax	<code>I2C1_Close();</code> or <code>I2C2_Close();</code> or <code>I2C3_Close();</code>
Arguments	None
Returns	None
Description	Calling this function closes the I ² C port and disables the I ² C hardware
Example	<code>I2C3_Close(); // Close I²C port and Disable the hardware</code>

2.12.3. I2C1_Start() or I2C2_Start() or I2C3_Start()

Syntax	I2C1_Start(); or I2C2_Start(); or I2C3_Start();	
Arguments	None	
Returns	Status (often ignored)	
	Status	1 if Successful 0 if Unsuccessful
Description	<p>Calling this function sends an I²C start condition. The hardware first pulls the SDA (data) line low, and next pulls the SCL (clock) line low.</p>  <p>The diagram shows two signals: SCL and SDA. SDA starts high, then drops to low. After a short delay, SCL also drops to low. Both signals remain low for a period before returning to high.</p>	
Example	I2C2_Start(); //Send an I ² C start condition.	

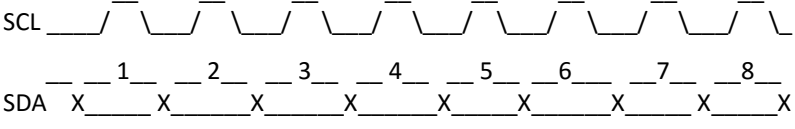
2.12.4. I2C1_Stop() or I2C2_Stop() or I2C3_Stop()

Syntax	<code>I2C1_Stop();</code> or <code>I2C2_Stop();</code> or <code>I2C3_Stop();</code>	
Arguments	None	
Returns	Status (often ignored)	
	Status	1 if Successful 0 if Unsuccessful
Description	<p>Calling this function sends an I²C stop condition. The hardware first releases the SCL to high state, and then releases the SDA line high.</p>  <p>The diagram shows two signals: SCL and SDA. SCL starts at a low level, then transitions to a high level. After SCL becomes high, SDA transitions from a low level to a high level. This sequence of events represents the hardware releasing the SCL line to high and then the SDA line to high, which is the condition for an I²C stop condition.</p>	
Example	<code>I2C1_stop(); // Send I²C Stop Condition</code>	

2.12.5. I2C1_Restart() or I2C2_Restart() or I2C3_Restart()

Syntax	I2C1_Restart(); or I2C2_Restart(); or I2C3_Restart();	
Arguments	None	
Returns	Status (often ignored)	
	Status	1 if Successful 0 if Unsuccessful
Description	Calling this function generates a restart condition.	
Example	I2C3_Restart() ; //Generates an I ² C restart condition	

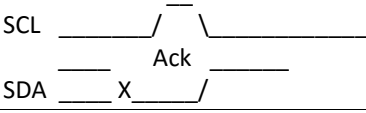
2.12.6. I2C1_Read() or I2C2_Read() or I2C3_Read()

Syntax	I2C1_Read(); or I2C2_Read(); or I2C3_Read();	
Arguments	None	
Returns	Byte	
	Byte	Byte from the I ² C Bus in the lower 8 bits.
Description	<p>Calling this function reads a single byte from the I²C bus. Note: Data can only change when the clock is low.</p>  <p>SCL _____</p> <p>_____ 1 _____ 2 _____ 3 _____ 4 _____ 5 _____ 6 _____ 7 _____ 8 _____</p> <p>SDA X _____ X _____ X _____ X _____ X _____ X _____ X _____ X _____</p>	
Example	ch := I2C1_Read() ; //Read a single byte from the I ² C Bus.	

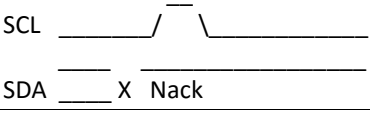
2.12.7. I2C1_Write(byte) or I2C2_Write(byte) or I2C3_Write(byte)

Syntax	I2C1_Write(byte); or I2C2_Write(byte); or I2C3_Write(byte);	
Arguments	byte	
	byte	The byte to be written to the I ² C Bus.
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Returns 2 if NACK received Returns 1 if ACK received Returns 0 if Failed
Description	Calling this function sends a single byte to the I ² C bus	
	<p>The diagram shows an I²C bus timing sequence. The top line is SCL (Serial Clock Line) and the bottom line is SDA (Serial Data Line). SCL has 8 clock pulses. SDA has 8 'X' marks, one under each SCL pulse, indicating that a byte of data is being transmitted. The SDA signal is high for the first 7 pulses and low for the 8th pulse, which is the ACK bit.</p>	
Example	Status := I2C3_Write(bytevalue); // Send a single byte to the I ² C	

2.12.8. I2C1_Ack() or I2C2_Ack() or I2C3_Ack()

Syntax	I2C1_Ack(); or I2C2_Ack(); or I2C3_Ack();
Arguments	None
Returns	None
Description	<p>Calling this function sends an I²C acknowledge condition. The hardware first pulls the SDA line low, and next releases SCL high followed by pulling SCL low again thus generating a clock pulse, SDA is then released high.</p> <p>NB:- Data can only change when the clock is low.</p>  <p>SCL _____/__________</p> <p> Ack</p> <p>SDA _____X_____</p>
Example	<code>I2C2_Ack(); // Send I²C Acknowledge condition</code>

2.12.9. I2C1_Nack() or I2C2_Nack() or I2C3_Nack()

Syntax	I2C1_Nack(); or I2C2_Nack(); or I2C3_Nack();
Arguments	None
Returns	None
Description	<p>Calling this function sends an I²C negative acknowledge condition. The hardware first release the SDA line high, and next releases SCL HI followed by pulling SCL low thus generating a clock pulse.</p> <p>NB:- Data can only change when the clock is low.</p>  <p>SCL _____</p> <p>SDA _____ X Nack</p>
Example	<pre>I2C3_Nack(); //Send an I²C Negative acknowledge condition</pre>

2.12.10. I2C1_AckStatus or I2C2_AckStatus or I2C3_AckStatus

Syntax	I2C1_AckStatus(); or I2C2_AckStatus(); or I2C3_AckStatus();	
Arguments	None	
Returns	Status	
	Status	Device Ack status
Description	<p>Call this function to get the ACK status from the slave device The state of SDA is returned.</p> <p>NB:- returns the state of SDA after the last clock pulse</p> <p>_____ Previous Clock Pulse</p> <p>SCL X _____</p> <p>SDA _____ X _____ Ack Status</p>	
Example	<pre>r := I2C1_AckStatus(); // returns the Ack Status.</pre>	

2.12.11. I2C1_AckPoll(control) or I2C2_AckPoll(control) or I2C3_AckPoll(control)

Syntax	I2C1_AckPoll(control); or I2C2_AckPoll(control); or I2C3_AckPoll(control);	
Arguments	control	
	control	The control word to be written to the device.
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Device Ack Status
Description	<p>Call this function to wait for a device to return an ACK during ACK polling The SDA is monitored for an Ack.</p> <p>NB:- returns the state of SDA after the last clock pulse</p> <p> ___ Previous Clock Pulse</p> <p>SCL X ___</p> <p>SDA ___ X ___ Ack Status</p>	
Example	<pre>r := I2C2_AckPoll(0xA0); //send the control byte the wait for a device //to return poll the device until an ACK //is received.</pre>	

2.12.12. I2C1_Idle() or I2C2_Idle() or I2C3_Idle()

Syntax	I2C1_Idle(); or I2C2_Idle(); or I2C3_Idle();	
Arguments	None	
Returns	Status	
	Status	1 if Successful 0 if Failed (Timed Out)
Description	<p>Call this function to wait until the I²C bus is inactive. NB:- wait for the bus to become idle. Times out if not inactive within 1 second.</p> <p>SCL X ___ X / _____ SDA X ___ X / _____</p>	
Example	r := I2C1_Idle(); //Wait until the I ² C Bus is inactive.	

2.12.13. I2C1_Gets(buffer, size) or I2C2_Gets(buffer, size) or I2C3_Gets(buffer, size)

Syntax	I2C1_Gets(buffer, size); or I2C2_Gets(buffer, size); or I2C3_Gets(buffer, size);	
Arguments	buffer, size	
	buffer	Storage for the string being read from the device.
	size	Maximum size of the string to be read
Returns	count	
	count	Returns the count of bytes actually read.
Description	Reads up to size characters into buffer from an ascii string stored in a device. Reads up to the ASCII NULL terminator and includes the terminator.	
Example	<pre>c := I2C3_Gets(buf, size); //read a string from the I²C Bus to buffer //up to size characters.</pre>	

2.12.14. I2C1_Getn() or I2C2_Getn() or I2C3_Getn()

Syntax	I2C1_Getn(buffer, count); or I2C2_Getn(buffer, count); or I2C3_Getn(buffer, count);	
Arguments	buffer, count	
	buffer	Storage for the bytes being read from the device.
	count	Number of bytes to be read
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Returns True if block read ok else returns False.
Description	Reads count bytes in to buffer and returns True if function succeeds	
Example	<pre>I2C1_Getn(buffer, count); //read I²C count bytes from the I2C Bus to //the buffer</pre>	

2.12.15. I2C1_Puts(buffer) or I2C2_Puts(buffer) or I2C3_Puts(buffer)

Syntax	I2C1_Puts(buffer); or I2C2_Puts(buffer); or I2C3_Puts(buffer);	
Arguments	buffer	
	buffer	Storage for the string being written to the device.
	The arguments can be a variable, array element, expression or constant	
Returns	Count	
	Count	Returns the count of bytes actually written.
Description	Writes an ASII string from buffer to a device. The ASCII NULL terminator is also written.	
Example	<pre>c := I2C3_Puts(mybuf); //write an ASCII string from buffer to the I²C //bus</pre>	

2.12.16. I2C1_Putn() or I2C2_Putn() or I2C3_Putn()

Syntax	I2C1_Putn(buffer, count); or I2C2_Putn(buffer, count); or I2C3_Putn(buffer, count);	
Arguments	buffer, count	
	buffer	Storage for the bytes being written to the device.
	count	Number of bytes to be written
Returns	written	
	written	Returns number of bytes written.
Description	Writes count bytes from the buffer to the device, and returns written if function succeeds.	
Example	<pre>b := I2C2_Putn(mybuf, count); // write count bytes from the buffer to // the I²C bus.</pre>	

2.13. Timer Functions

Summary of Functions in this section:

- `sys_T()`
- `sys_T_HI()`
- `sys_SetTimer(timernum, value)`
- `sys_GetTimer(timernum)`
- `sys_SetTimerEvent("timernum","function")`
- `sys_EventQueue()`
- `sys_EventsPostpone()`
- `sys_EventsResume()`
- `sys_DeepSleep(units)`
- `sys_Sleep(units)`
- `iterator(offset)`
- `sys_GetDate()`
- `sys_GetTime()`
- `sys_SetDate(year, month, day)`
- `sys_SetTime(hours, mins, secs)`
- `sys_GetDateVar(&year, &month, &day)`
- `sys_GetTimeVar(&hour, &minute, &second, &msecs)`

2.13.1. sys_T()

Syntax	<code>sys_T();</code>	
Arguments	None	
Returns	value	
	value	Returns the value of system timer. (LO Word)
Description	Returns the current value of the rolling 32bit system timer (1msec) LO word.	
Example	<code>t := sys_T(); // .</code>	

2.13.2. sys_T_HI()

Syntax	<code>sys_T_HI();</code>	
Arguments	None	
Returns	value	
	value	Returns the value of system timer. (HI Word)
Description	Returns the current value of the rolling 32bit system timer (1mse) HI word.	
Example	<code>t := sys_T_HI(); //</code>	

2.13.3. sys_SetTimer(timernum, value)

Syntax	<code>sys_SetTimer(timernum, value);</code>	
Arguments	timernum, value	
	timernum	One of eight timers TIMER0 to TIMER7.
	value	Countdown period in milliseconds.
	The "value" can be a variable, array element, expression or constant	
Returns	None	
Description	Set a countdown on the selected timer or 'top-up' if required. There are 8 timers TIMER0 to TIMER7 which stop at the count of 0. Maximum timeout period is 65, 535 milliseconds or 65.535 seconds. A timer can be read with the <code>sys_GetTimer("timernum")</code> function.	
Example	<code>sys_SetTimer(TIMER5, 3600); //Set Timer5 for 1 hour.</code>	

2.13.4. sys_GetTimer(timernum)

Syntax	<code>sys_GetTimer(timernum);</code>	
Arguments	timernum	
	timernum	One of eight timers TIMER0 to TIMER7.
Returns	Value	
	Value	Returns 0 if timer has expired, or the current countdown value.
Description	Returns 0 if timer has expired, or the current countdown value. There are 8 timers TIMER0 to TIMER7 which stop at the count of 0. Maximum timeout period is 65, 535 milliseconds or 65.535 seconds. A timer can be set with the <code>sys_SetTimer("timernum", "value")</code> function.	
Example	<code>t := sys_GetTimer(TIMER2); //</code>	

2.13.5. sys_SetTimerEvent(timernum, function)

Syntax	<code>sys_SetTimerEvent(timernum, function);</code>	
Arguments	timernum, function	
	timernum	One of eight timers TIMER0 to TIMER7.
	function	Event Function to be queued
Returns	Address	
	Address	Returns any previous event function address, or zero if there was no previous function.
Description	<p>Set a function to be called for selected timer. When the timer reaches zero, the function is called. The called function must not have any parameters, and should not have a return value. This is necessary because the timer event is invoked asynchronously to the mainline program (i.e, it is not called in the normal way, so parameters and return values don't apply).</p> <p>Note: When a child process is run using the file_run or file_exec function, or if a file was loaded with file_Loadfunction and is executed, the loaded process gets its own code and memory space, therefore, any timer that reaches zero that has a timer event attached in the parent code space, will fail and cause a crash as an attempt is made to force the program counter to some wild place in the child process - There are 2 ways to overcome this problem.</p> <p>1] If a child process will not be requiring the use of any timers or timer events, the parent program can simply use the eventsPostpone() function before calling or entering the child process. Once the parent program regains control, the eventsResume() function will allow any events in the queue to then be processed. The side effect of this method is that several events may bank up, and will execute immediately once the eventsResume() takes place. This however disallows a child process to use any timer events in the sub program so method 2 is preferable in this case.</p> <p>2] The parent program can 'disconnect' the event(s) by setting it/them to zero prior to child process execution, or setting the associated timer to zero so the event wont fire. In either case, it is necessary to do the following:- <pre>while(sys_EventQueue());</pre> to ensure the event queue is empty prior to calling the child process. Note also that if just the timer is set to zero, the child process cannot use this timer. If the timer was now set to a value and the old event still existed, when the timer reaches zero the 'bad' parent address event will fire causing a crash.</p> <p>The reverse situation also applies of course, the same level of respect is required if a child program needs to use any timer events. Method [1] (above) will not work as the events have been postponed, stopping the child process from using any timer events. If the child process did an eventsResume() in this case, everything would crash miserably. So the same applies, a child that uses any timer events must respect any timers that may be used by the parent, and a child must zero the sys_SetTimerEvent before returning to the parent.</p> <p>sys_SetTimerEvent(timernum, 0) disables the timer event.</p>	
Example	<code>sys_SetTimerEvent (TIMER5, myfunc) ;</code>	

2.13.6. sys_EventQueue()

Syntax	<code>sys_EventQueue();</code>	
Arguments	None	
Returns	Count	
	Count	Returns number of events .
Description	returns the max number of events that were pending in the event queue since the last call to this function. This can be used to assess event overhead burden, especially after or during a <code>sys_EventsPostpone</code> action..	
Example	<code>tasks := sys_EventQueue(); //</code>	

2.13.7. sys_EventsPostpone()

Syntax	<code>sys_EventsPostpone();</code>
Arguments	None
Returns	None
Description	<p>Postpone any events until the <code>sys_EventResume</code> function is executed. The event queue will continue to queue events, but no action will take place until a <code>sys_EventResume</code> function is encountered.</p> <p>The queue will continue to receive up to 32 events before discarding any further events. This function is required to allow a sequence of instructions or functions to occur that would otherwise be corrupted by an event occurring during the sequence of instructions or functions. A good example of this is when you set a position to print, if there was no way of locking the current sequence, an event may occur which does a similar thing, and a contention would occur - printing to the wrong position. This function should be used wisely, if any action that is required would take considerable time, it is better to disable any conflicting event functions with a bypass flag, then restart the conflicting event by re-issuing a timer value.</p>
Example	<code>sys_EventsPostpone(); // postpone the event queue</code>

2.13.8. sys_EventsResume()

Syntax	<code>sys_EventsResume();</code>
Arguments	None
Returns	None
Description	Resume any postponed events. The queue will try to execute any events that were incurred during the postponed period. Note that queued events are only checked for and executed at the the end of each 4DGL instruction.
Example	<code>sys_EventsResume(); // resume the event queue</code>

2.13.9. sys_DeepSleep(units)

Syntax	<code>sys_DeepSleep(units);</code>	
Arguments	units	
	units	Sleep timer units are approx 1 second. When in sleep mode, timing is controlled by an RC oscillator, therefore, timing is not totally accurate and should not be relied on for timing purposes
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Remaining time units when touch screen is touched, else returns zero.
Description	Put the display and processor into lowest power mode for a period of time. If "units" is zero, the display goes into sleep mode forever and needs power cycling to re-initialize. If "units" is 1 to 65535, the display will sleep for that period of time, or will be woken when touch screen is touched. The function returns the count of "units" that are remaining when the screen was touched. When returning from deep sleep mode, the processor is restored from low power mode, the display should be reinitialised with <code>disp_Init()</code> . New in v0.7 PmmC	
Example	<code>sys_DeepSleep(60); // Sleep for 1 minute.</code>	

2.13.10. sys_Sleep(units)

Syntax	<code>sys_Sleep(units);</code>	
Arguments	units	
	units	Sleep timer units are approx 1 second. When in sleep mode, timing is controlled by an RC oscillator, therefore, timing is not totally accurate and should not be relied on for timing purposes
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Remaining time units when touch screen is touched, else returns zero.
Description	Put the display and processor into low power mode for a period of time. If "units" is zero, the display goes into sleep mode forever and needs power cycling to re-initialize. If "units" is 1 to 65535, the display will sleep for that period of time, or will be woken when touch screen is touched. The function returns the count of "units" that are remaining when the screen was touched. When returning from sleep mode, the display and processor are restored from low power mode. Note: Sys_Sleep() was found to have an issue in PmmC's prior to R33, the units value was not always near 1 second. This has been corrected in PmmC R33.	
Example	<code>sys_Sleep(60); // Sleep for 1 minute.</code>	

2.13.11. iterator(offset)

Syntax	<code>iterator_(offset);</code>	
Arguments	offset	
	offset	Offset size for the next ++ or -- command
	The arguments can be a variable, array element, expression or constant	
Returns	None	
Description	Sets the iterator size for the next postinc, postdec, preinc or predec by a specified value. The offset will return to 1 after the next operation.	
Example	<code>t := iterator(10); // Set the iterator size to be 10</code>	

2.13.12. sys_GetDate()

Syntax	<code>sys_GetDate();</code>
Arguments	None
Returns	None
Description	Print the system date in the format "DD-MM-YYYY" Can be captured to a buffer using the to() function.
Example	<code>Sys_GetDate(); // Print the current Date to the display</code>

2.13.13. sys_GetTime()

Syntax	<code>sys_GetTime();</code>
Arguments	None
Returns	None
Description	Print the system time in the format "HH:MM:SS" Can be captured to a buffer using the to() function.
Example	<pre>var buf[5]; to(buf); Sys_GetTime(); // Print the current Time to the buffer</pre>

2.13.14. sys_SetDate(year, month, day)

Syntax	<code>sys_SetDate(year, month, day);</code>	
Arguments	<code>year, month, day</code>	
	year	Year argument can be a variable, array element, expression or constant
	month	Month argument can be a variable, array element, expression or constant
	day	Day argument can be a variable, array element, expression or constant
Returns	Status	
	Status	TRUE if valid date
Description	<p>Used to set clock to correct date after power up or suspension.</p> <p>If an I2C real time clock is present, this function can be used to synchronize the internal date to the I2C RTC date.</p> <p>Returns true if valid date.</p>	
Example	<code>Sys_SetDate(13, 08, 05);</code>	

2.13.15. `sys_SetTime(hour, minute, second)`

Syntax	<code>sys_SetTime(hour, minute, second);</code>	
Arguments	hour, minute, second	
	hour	Hour argument can be a variable, array element, expression or constant
	minute	Minute argument can be a variable, array element, expression or constant
	second	Second argument can be a variable, array element, expression or constant
Returns	Status	
	Status	TRUE if valid time
Description	<p>Used to set clock to correct time after power up or suspension.</p> <p>If an I2C real time clock is present, this function can be used to synchronize the internal time to the I2C RTC time.</p> <p>Returns true if valid time.</p>	
Example	<code>Sys_SetTime(11, 03, 55);</code>	

2.13.16. sys_GetDateVar(&year, &month, &day)

Syntax	<code>sys_GetDateVar(&year, &month, &day);</code>	
Arguments	<code>year, month, day</code>	
	<code>year</code>	Specifies the address for the storage location of the returned year value
	<code>month</code>	Specifies the address for the storage location of the returned month value
	<code>day</code>	Specifies the address for the storage location of the returned day value
Returns	None	
Description	Returns the current year, month and day into variables.	
Example	<pre>Sys_GetDateVar(&year, &month, &day); // Read the current Date into variables</pre>	

2.13.17. `sys_GetTimeVar(&hour, &minute, &second, &msecs)`

Syntax	<code>sys_GetTimeVar(&hour, &minute, &second, &msecs);</code>	
Arguments	hour, minute, second, msecs	
	hour	Specifies the address for the storage location of the returned hour value
	minute	Specifies the address for the storage location of the returned minute value
	second	Specifies the address for the storage location of the returned second value
	msecs	Specifies the address for the storage location of the returned milli-second value
Returns	None	
Description	Returns the current hour, minute, second and milli-second into variables.	
Example	<pre> Sys_GetTimeVar(&hour, &minute, &second, &msecs); // Get the current Time into variables </pre>	

2.14. FAT16 File Functions

Summary of Functions in this section:

- file_Error()
- file_Count(filename)
- file_Dir(filename)
- file_FindFirst(fname)
- file_FindNext()
- file_Exists(fname)
- file_Open(fname, mode)
- file_Close(handle)
- file_Read(destination, size, handle)
- file_Seek(handle, HiWord, LoWord)
- file_Index(handle, Hisize, Losize, recordnum)
- file_Tell(handle, &HiWord, &LoWord)
- file_Write(Source, size, handle)
- file_Size(handle, &HiWord, &LoWord)
- file_Image(x, y, handle)
- file_ScreenCapture(x, y, width, height, handle)
- file_PutC(char, handle)
- file_GetC(handle)
- file_PutW(word, handle)
- file_GetW(handle)
- file_PutS(source, handle)
- file_GetS(*String, size, handle)
- file_Erase(fname)
- file_Rewind(handle)
- file_LoadFunction(fname.4XE)
- file_Run(fname.4XE, arglistptr)
- file_Exec(fname.4XE, arglistptr)
- file_LoadImageControl(fname1, fname2, mode)
- file_Mount()
- file_Unmount()
- file_PlayWAV
- file_Rename(oldname, newname)
- file_SetDate(handle, year, month, day, hour, minute, second)

2.14.1. file_Error()

Syntax	file_Error();		
Arguments	None.		
Returns	Error Code		
	ERROR CODE	ERROR NO.	ERROR DESCRIPTION
	FE_OK	0	IDE function succeeded
	FE_IDE_ERROR	1	IDE command execution error
	FE_NOT_PRESENT	2	CARD not present
	FE_PARTITION_TYPE	3	WRONG partition type, not FAT16
	FE_INVALID_MBR	4	MBR sector invalid signature
	FE_INVALID_BR	5	Boot Record invalid signature
	FE_DISK_NOT_MNTD	6	Media not mounted
	FE_FILE_NOT_FOUND	7	File not found in open for read
	FE_INVALID_FILE	8	File not open
	FE_FAT_EOF	9	Fat attempt to read beyond EOF
	FE_EOF	10	Reached the end of file
	FE_INVALID_CLUSTER	11	Invalid cluster value > maxcls
	FE_DIR_FULL	12	All root dir entry are taken
	FE_DISK_FULL	13	All clusters in partition are taken
	FE_FILE_OVERWRITE	14	A file with same name exist already
	FE_CANNOT_INIT	15	Cannot init the CARD
	FE_CANNOT_READ_MBR	16	Cannot read the MBR
	FE_MALLOC_FAILED	17	Malloc could not allocate the FILE struct
	FE_INVALID_MODE	18	Mode was not r.w.
	FE_FIND_ERROR	19	Failure during FILE search
	FE_INVALID_FNAME	20	Invalid Filename
	FE_INVALID_MEDIA	21	bad media
	FE_SECTOR_READ_FAIL	22	Sector Read fail
	FE_SECTOR_WRITE_FAIL	23	Sector write fail
Description	Returns the most recent error code.		
Example	<code>e := file_Error(); // File Error</code>		

2.14.2. file_Count(filename)

Syntax	<code>file_Count(filename);</code>	
Arguments	filename	
	filename	Name of the file(s) for the search (passed as a string)
Returns	Count	
	Count	Number of files that match the criteria.
Description	Returns number of files found that match the criteria. The wild card character '*' matches up with any combination of allowable characters and '?' matches up with any single allowable character.	
Example	<code>count := file_Count("*.4XE"); //Returns number of files with ".4XE".</code>	

2.14.3. file_Dir(filename)

Syntax	<code>file_Dir(filename);</code>	
Arguments	filename	
	filename	Name of the file(s) for the search (passed as a string)
Returns	Count	
	Count	Number of files found that match the criteria.
Description	Streams a string of file names that agree with the search key. Returns number of files found that match the criteria. The wild card character '*' matches up with any combination of allowable characters and '?' matches up with any single allowable character.	
Example	<code>count := file_Dir("*.4XE"); //Returns number of files with ".4XE".</code>	

2.14.4. file_FindFirst(fname)

Syntax	<code>file_FindFirst(fname);</code>	
Arguments	fname	
	fname	Name of the file(s) for the search (passed as a string)
Returns	Status	
	Status	1: If at least one file exists that satisfies the criteria. 0: If no file satisfies the criteria.
Description	Returns true if at least 1 file exists that satisfies the file argument. Wildcards are usually used so if file_FindFirst returns true, further tests can be made using file_FindNext(); to find all the files that match the wildcard class. Note that the stream behaviour is the same as file_Dir.	
Example	<pre>If (file_FindFirst (*.4XE)) Print("File Found"); endif</pre>	

2.14.5. file_FindNext()

Syntax	<code>file_FindNext();</code>	
Arguments	None	
Returns	Status	
	Status	1: If more files exist that satisfy the criteria set in the <code>file_FindFirst(fname)</code> 0: If no more files satisfy the criteria set in the <code>file_FindFirst(fname)</code>
Description	Returns true if more file exists that satisfies the file argument that was given for <code>file_FindFirst</code> . Wildcards must be used for <code>file_FindFirst</code> , else this function will always return zero as the only occurrence will have already been found. Note that the stream behaviour is the same as <code>file_Dir</code> .	
Example	<pre>while ((file_FindNext()) filecount++; wend</pre>	

2.14.6. file_Exists(fname)

Syntax	<code>file_Exists(fname);</code>	
Arguments	<code>fname</code>	
	<code>fname</code>	Name of the file for the search (passed as a string)
Returns	Status	
	Status	1: File found 0: File not found
Description	Tests for the existence of the file provided with the search key. Returns TRUE if found.	
Example	<pre>If (file_Exists("fill.4XE")) Print("File Found"); endif</pre>	

2.14.7. file_Open(fname, mode)

Syntax	<code>file_Open(fname, mode);</code>	
Arguments	fname, mode	
	fname	Name of the file to be opened (passed as a string)
	mode	FILE_READ: 'r' FILE_WRITE: 'w' FILE_APPEND: 'a'
Returns	handle	
	handle	Returns handle if file exists. Sets internal file error number accordingly (0 if no errors).
Description	<p>Returns handle if file exists. The file "handle" that is created is now used as reference for "filename" for further file functions such as file_Close(handle), etc. For FILE_WRITE and FILE_APPEND modes ('w' and 'a') the file is created if it does not exist. If the file is opened for append and it already exists, the file pointer is set to the end of the file ready for appending, else the file pointer will be set to the start of the newly created file.</p> <p>If the file was opened successfully, the internal error number is set to 0 (i.e. no errors) and can be read with the file_Error() function..</p> <p>For FILE_READ mode ('r') the file must exist else a null handle (0) is returned and the 'file not found' error number is set which can be read with the file_Error() function..</p> <p>Note: If a file is opened for write mode 'w', and the file already exists, the operation will fail. Unlike C and some other languages where the file will be erased ready for re-writing when opened for writing, 4DGL offers a simple level of protection that ensures that a file must be purposely erased before being re-written.</p> <p>Note: Beginning with the v4.0 PmmC a file opened with FILE_APPEND may be randomly read and or written. Also any altered file will have the Archive bit set in the directory entry.</p>	
Example	<code>handle := file_Open("myfile.txt", 'r');</code>	

2.14.8. file_Close(handle)

Syntax	<code>file_Close(handle);</code>	
Arguments	handle	
	handle	the file handle that was created by <code>file_Open("fname")</code> which is now used as reference (handle) for "fname" for further file functions such as in this function to close the file.
Returns	Status	
	Status	1: File Closed. 0: File not closed.
Description	Returns TRUE if file closed, FALSE if not.	
Example	<code>res := file_Close(hndl);</code>	

2.14.9. file_Read(destination, size, handle)

Syntax	<code>file_Read(*destination, size, handle);</code>	
Arguments	destination, size, handle	
	destination	Destination memory buffer
	size	Number of bytes to be read
	handle	The handle that references the file to be read.
Returns	count	
	count	Returns the number of characters read.
Description	<p>Reads the number of bytes specified by "size" from the file referenced by "handle" into a destination memory buffer.</p> <p>If "destination" is zero, data is read direct to GRAM window</p>	
Example	<code>res := file_Read(memblock, 20, hnd11);</code>	

2.14.10. file_Seek(handle, HiWord, LoWord)

Syntax	<code>file_Seek(handle, HiWord, LoWord);</code>	
Arguments	handle, HiWord, LoWord	
	handle	The handle that references the file
	HiWord	Contains the upper 16bits of the memory pointer into the file
	LoWord	Contains the lower 16bits of the memory pointer into the file
Returns	Status	
	Status	Returns TRUE if ok, usually ignored
Description	Places the file pointer at the required position in a file that has been opened in 'r' (read) or 'a' (append) mode. In append mode, file_Seek does not expand a filesize, instead, the file pointer (handle) is set to the end position of the file, eg:- assuming the file size is 10000 bytes, file_Seek(handle, 0, 0x1234); will set the file position to 0x00001234 (byte position 4660) for the file handle, so subsequent data may be read from that position onwards with file_GetC(...), file_GetW(...), file_GetS(...), or an image can be displayed with file_Image(...). Conversely, file_PutC(...), file_PutW(...) and file_PutS(...) can write to the file at the position. A FE_EOF (end of file error) will occur if you try to write or read past the end of the file.	
Example	<code>res := file_Seek(hSource, 0x0000, 0x1234) ;</code>	

2.14.11. file_Index(handle, Hisize, LoSize, recordnum)

Syntax	<code>file_Index(handle, Hisize, LoSize, recordnum);</code>	
Arguments	<code>handle, Hisize, LoSize, recordnum</code>	
	handle	The handle that references the file
	Hisize	Contains the upper 16bits of the size of the file records.
	LoSize	Contains the lower 16bits of the size of the file records.
	recordnum	The index of the required record
Returns	Status	
	Status	Returns TRUE if ok, usually ignored
Description	Places the file pointer at the position in a file that has been opened in 'r' (read) or 'a' (append) mode. In append mode, file_Index does not expand a filesize, instead, the file pointer (handle) is set to the end position of the file, eg:- assuming the record size is 100 bytes, file_Index(handle, 0, 100, 22); will set the file position to 2200 for the file handle, so subsequent data may be read from that position onwards with file_GetC(...), file_GetW(...), file_GetS(...), or an image can be displayed with file_Image(...). Conversely, file_PutC(...), file_PutW(...) and file_PutS(...) can write to the file at the position. A FE_EOF (end of file error) will occur if you try to write or read past the end of the file.	
Example	<code>res := file_Index(hSource, 0, 100, 22) ;</code>	

2.14.12. file_Tell(handle, &HiWord, &LoWord)

Syntax	<code>file_Tell(handle, &HiWord, &LoWord);</code>	
Arguments	handle, &HiWord, &LoWord	
	handle	The handle that references the file
	HiWord	Contains the upper 16bits of the memory pointer into the file
	LoWord	Contains the lower 16bits of the memory pointer into the file
Returns	Status	
	Status	Returns TRUE if ok, usually ignored
Description	Reads the 32 bit file pointer and stores it into 2 variables, HiWord and LoWord	
Example	<code>res := file_Tell(hSource, &HIptr, &LOptr) ;</code>	

2.14.13. file_Write(*source, size, handle)

Syntax	<code>file_Write(*source, size, handle);</code>	
Arguments	source, size, handle	
	source	Source memory buffer.
	size	Number of bytes to be written.
	handle	The handle that references the file to write.
Returns	count	
	count	Returns the number of bytes written.
Description	Writes the number of bytes specified by "size" from the source buffer into the file referenced by "handle".	
Example	<code>res := file_Write(memblock, 20, hndl1);</code>	

2.14.14. file_Size(handle, &HiWord, &LoWord)

Syntax	<code>file_Size(handle, &HiWord, &LoWord);</code>	
Arguments	handle, HiWord, LoWord	
	handle	The handle that references the file.
	HiWord	Contains the upper 16bits of the file size.
	LoWord	Contains the lower 16bits of the file size.
Returns	Status	
	Status	Returns TRUE if ok, usually ignored.
Description	Reads the 32 bit file size and stores it into 2 variables, HiWord and LoWord	
Example	<code>res := file_Size(hSource, &sizeHi, &sizeLo);</code>	

2.14.15. file_Image(x, y, handle)

Syntax	<code>file_Image(x, y, handle);</code>	
Arguments	<code>x, y, handle</code>	
	x	X-position of the image to be displayed
	y	Y-position of the image to be displayed
	handle	The handle that references the file containing the image(s)
Returns	Returns a copy of the file_Error() error code	
Description	Display an image from the file stream at screen location specified by x, y(top left corner). If there is more than 1 image in the file, it can be accessed with file_Seek(...).	
Example	<code>file_Image(x, y, handle) ;</code>	

2.14.16. file_ScreenCapture(x, y, width, height, handle)

Syntax	<code>file_ScreenCapture(x, y, width, height, handle);</code>	
Arguments	x, y, width, height, handle	
	x	X-position of the image to be captured
	y	Y-position of the image to be captured
	width	Width of the area to be captured.
	height	Height of the area to be captured.
	handle	The handle that references the file to store the image(s)
Returns	Status	
	Status	Returns 0 if function successful.
Description	<p>Save an image of the screen shot to file at the current file position. The image can later be displayed with <code>file_Image(...)</code>; The file may be opened in append mode to accumulate multiple images. Later, the images can be displayed with <code>file_Seek(...)</code>.</p> <p>Note that the image will be sector aligned.</p> <p>All image headers must start on a sector boundary.</p> <p>The image is saved from x, y (with respect to top left corner), and the capture area is determined by "width" and "height".</p>	
Example	<pre>file_Mount(); hFile := file_Open("test.img", 'a'); // open a file to save the image file_ScreenCapture(20,20,100,100, hFile); // save an area file_ScreenCapture(0,0,50,50, hFile); // (save another area) file_Close(hFile); // now close the file // and to display the saved area(s) hFile := file_Open("test.img", 'r'); // open the saved file file_Image(20,180, hFile); // display the image file_Image(150,180, hFile); // (display the next image) file_Close(hFile); file_Unmount(); // finished with file system</pre>	

2.14.17. file_PutC(char, handle)

Syntax	<code>file_PutC(char, handle);</code>	
Arguments	<code>char, handle</code>	
	char	Data byte about to be written.
	handle	The handle that references the file to be written to.
Returns	Status	
	Status	Returns true if function succeeded
Description	This function writes the byte specified by "char" to the file, at the position indicated by the associated file-position pointer and advances the pointer appropriately (incremented by 1). The file must be previously opened with 'w' (write) or 'a' (append) modes.	
Example	<code>file_PutC('A', hndl);</code>	

2.14.18. file_GetC(handle)

Syntax	<code>file_GetC(handle);</code>	
Arguments	handle	
	handle	The handle that references the file.
Returns	byte	
	byte	Returns the next char from the file
Description	This function reads a byte from the file, at the position indicated by the associated file-position pointer and advances the pointer appropriately (incremented by 1). The file must be previously opened with 'r' (read) mode.	
Example	<code>mychar := file_GetC(hndl) ;</code>	

2.14.19. file_PutW(word, handle)

Syntax	<code>file_PutW(word, handle);</code>	
Arguments	word, handle	
	word	Data about to be written
	handle	The handle that references the file to be written to.
Returns	Status	
	Status	Returns true if function succeeded
Description	This function writes word sized (2 bytes) data specified by " word " to the file, at the position indicated by the associated file-position pointer and advances the pointer appropriately (incremented by 2). The file must be previously opened with 'w' (write) or 'a' (append) modes.	
Example	<code>file_PutW(0x1234, hndl);</code>	

2.14.20. file_GetW(handle)

Syntax	<code>file_GetW(handle);</code>	
Arguments	handle	
	handle	The handle that references the file.
Returns	Word	
	Word	Returns the next word in the file
Description	This function reads a word (2 bytes) from the file, at the position indicated by the associated file-position pointer and advances the pointer appropriately (incremented by 2). The file must be previously opened with 'r' (read) mode.	
Example	<code>myword := file_GetW(hndl);</code>	

2.14.21. file_PutS(*source, handle)

Syntax	<code>file_PutS(*source, handle);</code>	
Arguments	<code>source, handle</code>	
	source	A pointer to the string to be written.
	handle	The handle that references the file to be written to.
Returns	<code>count</code>	
	count	Returns the number of characters written (excluding the null terminator).
Description	This function writes an ASCIIZ (null terminated) string from a buffer specified by " *source " to the file, at the position indicated by the associated file-position pointer and advances the pointer appropriately. The file must be previously opened with 'w' (write) or 'a' (append) modes.	
Example	<code>file_PutS(mystring, hndl);</code>	

2.14.22. file_GetS(*string, size, handle)

Syntax	<code>file_GetS(*string, size, handle);</code>	
Arguments	<code>string, size, handle</code>	
	string	Destination buffer
	size	The maximum number of bytes to be read from the file. (Up to max of 80)
	handle	The handle that references the file.
Returns	Count	
	Count	Returns the number of characters read from file (excluding the null terminator)
Description	<p>This function reads a line of text to a buffer (specified by "*string") from a file at the current file position indicated by the associated file-position pointer and advances the pointer appropriately. The file must be previously opened with 'r' (read) mode.</p> <p>Note: only reads up to "size-1" characters into "string"</p> <p>file_GetS(...) will stop reading when any of the following conditions are true:</p> <ul style="list-style-type: none"> A) It has read n-1 bytes (one character is reserved for the null-terminator) B) It encounters a newline character (a line-feed in the compilers tested here) C) It reaches the end of file D) A read error occurs. <p>The file must be previously opened with 'r' (read) mode.</p>	
Example	<code>res := file_GetS(mystring, 80, hndl);</code>	

2.14.23. file_Erase(fname)

Syntax	<code>file_Erase(fname);</code>	
Arguments	<code>fname</code>	
	<code>fname</code>	Name of the file to be erased
Returns	Status	
	Status	1: if successful 0: if unsuccessful
Description	This function erases a file on the disk. Note: If the function fails, the appropriate error number is set in <code>file_Error()</code> and will usually be error 19, "failure during FILE search".	
Example	<code>res := file_Erase("myfile.txt") ;</code>	

2.14.24. file_Rewind(handle)

Syntax	<code>file_Rewind(handle);</code>	
Arguments	handle	
	handle	The handle that references the file
Returns	Status	
	Status	Returns TRUE if ok, usually ignored
Description	Resets the file pointer to the beginning of a file that has been opened in 'r' (read), 'w', or 'a' (append) mode.	
Example	<pre>res := file_Rewind(hSource);</pre>	

2.14.25. file_LoadFunction(fname.4XE)

Syntax	<code>file_LoadFunction(fname.4XE);</code>	
Arguments	<code>fname.4XE</code>	
	<code>fname.4XE</code>	Name of the 4DGL application program that is about to be loaded into RAM.
Returns	Pointer	
	Pointer	Returns a pointer to the memory allocation where the function has been loaded from file which can be then used as a function call.
Description	<p>Load a function or program from disk and return a function pointer to the allocation. The function can then be invoked just like any other function would be called via a function pointer. Parameters may be passed to it in a conventional way. The function may be discarded at any time when no longer required, thus freeing its memory resources.</p> <p>The loaded function can be discarded with <code>mem_Free(..)</code> Note that any pointer references passed to the child function may not include references to the parents DATA statements or any static string references. Any string or array information must be in the parents global or local memory space. The reason for this is that DATA statements and static strings are contained in the parents CODE segment, and cannot be accessed by the child process.</p> <p>The callers stack is shared by the loaded function, however any global variables in the loaded function are private to that function.</p>	
Example1	<pre>var titlestring[20]; var textstring[20]; to(titlestring); putstr("My Window Title"); to(textstring); putstr("My Special Message"); popupWindow := file_LoadFunction("popupWindow1.4fn"); if(!popupWindow)goto LoadFunctionFailed; //could not load the function //then elsewhere in your program res := popupWindow(MYMODE,titlestring,textstring); if(res == QUIT_APPLICATION) goto exitApp; //Later in your program, when popupWindow is no longer required //for the application res := mem_Free(popupWindow); if(!res) goto FreeFunctionFailed; //should never happen if memory not //corrupted</pre>	
Example2	<pre>var fncHandle; //a var for a handle to sliders2.4dg var slidervals; //reference var to access global vars in sliders.4dg fncHandle := file_LoadFunction("sliders2.4xe"); // load the function slidervals := fncHandle&0x7FFF; // note that memory allocations for transient programs are biased with 8000h which must be removed. slidervals++; // note that all globals start at '1' slidervals[0] := 25; // set sliders to initial positions slidervals[1] := 20; slidervals[2] := 30; slidervals[3] := 15; slidervals[4] := 35; slidervals[5] := 20;</pre>	

```
slidervals[6] := 40;
slidervals[7] := 25;
slidervals[8] := 45;
slidervals[9] := 5;

r := fncHandle(); // activate the function

print("Return value = 0x", [HEX] r, "\n");

// print the values, they may have changed
print("Slider 1 ", slidervals[0], " Slider 2 ", slidervals[1], "\n");
print("Slider 3 ", slidervals[2], " Slider 4 ", slidervals[3], "\n");
print("Slider 5 ", slidervals[4], " Slider 6 ", slidervals[5], "\n");
print("Slider 7 ", slidervals[6], " Slider 8 ", slidervals[7], "\n");
print("Slider 9 ", slidervals[8], " Slider 10 ", slidervals[9], "\n");

mem_Free(fncHandle); // done with sliders, release its memory
```

2.14.26. file_Run(fname.4XE, arglistptr)

Syntax	file_Run(fname.4XE, arglistptr);	
Arguments	fname.4XE, arglistptr	
	fname.4XE	name of the 4DGL child program to be loaded into RAM and executed.
	arglistptr	pointer to the list of arguments to pass to the new program.
Returns	Value	
	Value	Returns the value from main in the called program.
Description	<p>Any memory allocations in the main FLASH program are released, however, the stack and globals are maintained.</p> <p>If arglistptr is 0, no arguments are passed, else arglistptr points to an array, the first element being the number of additional elements in the array which contain the arguments.</p> <p>func 'main' in the called program accepts the arguments, if any.</p> <p>The arguments can only be passed by value, no pointers or references can be used as all memory is cleared before the file is loaded. Refer to file_Exec and file_LoadFunction for functions that can pass by reference.</p> <p>The disk does not need to be mounted, file_Run automatically mounts the drive.</p>	
Example	<pre>#inherit "4DGL_16bitColours.fnc" #inherit "FONT4.fnt" #constant MAXBUTTONS 30 // for now, maximum number of buttons we want // (also sets maximum number of files we can exec) #STACK 500 //stack must be large enough to be shared with called program #MODE RUNFLASH // This is a 'top down' main program and must be run from FLASH //-----// local global variables //----- // NB:- demo assigns all arrays to MAXBUTTONS. // The arrays could be dynamically assigned to minimise memory usage. // There is break even point between extra code and smallish arrays. var keyval; // 0 if no key pressed else 1-n var filenames; // pointer to byte array that holds the filenames var buttontexts[MAXBUTTONS]; // pointers into the filenames array //holds the filenames we use as button text var vButtonState[MAXBUTTONS]; //button state flag(bit 0 = up:down state) var vOldButtonState[MAXBUTTONS]; // OLD button state flags (bit 0 = up:down state) // (we keep 2 copies so we can test for a state change and only redraw when a state change occurs) var touchX1[MAXBUTTONS]; // touch regions for the buttons</pre>	

```

var touchY1[MAXBUTTONS];
var touchX2[MAXBUTTONS];
var touchY2[MAXBUTTONS];

var btnTextColor;           // button text colour
var btnBtnColor;           // button background colour
var buttoncount;           // actual number of buttons created
                             (set by number of *.4XE files we find on drive)

var tempstr[20];           // general purpose string, 40 bytes

#DATA
  byte fred 1,2,3,4,5,6,7,8,9,10,11,12
#END

/*=====
Redraw the button matrix. Only draw buttons that have changed state.
The top left corner of the button matrix is set with the xorg and yorg
parameters depending on the font and text string width, the button
matrix dynamically resizes.
Parameters:-
maxwidth    = rhs from xorg (in pixels) to cause wrap at rhs
maxwidth    = maximum matrix width (in pixel units)
buttoncount = number of buttons to display
font        = FONT_1 to FONT_4
xorg:yorg   = top left corner of button array
NB:- The touch detect matrix array is updated when any button changes
state.
When you need to draw the matrix for the first instance of the matrix,
you must
call with mode = 1 to instantiate the buttons.
call with mode = 0 for normal button action.
=====*/

func redraw(var bcount, var font, var xorg, var yorg, var maxwidth, var
mode )

  var xgap, ygap, n, x1, y1, x2, y2;

  xgap := 2;
  ygap := 2;
  x1 := xorg;
  y1 := yorg;

  // if first, set all the buttons to the up state
  if (mode)
    n := 0;
    repeat
      vButtonState[n]:=UP;
    // set all the buttons to inverse state
      vOldButtonState[n]:=DOWN;
    // so we guarantee they are all drawn in the 'up' state (not pressed)
      until(++n >= buttoncount);
    endif

  // check all the button states, if a change occurred, draw the new
  button state and update the touch detect matrix array
  n := 0;
  repeat
    // if the button state has changed
    if ( vButtonState[n] != vOldButtonState[n])
      vOldButtonState[n] := vButtonState[n];

    // if we already have all the co-ordinates, use them
    if (!mode)
      x1 := touchX1[n];
      y1 := touchY1[n];
      x2 := touchX2[n];

```

```

        y2 := touchY2[n];
    endif

    // draw the button
    gfx_Button( vButtonState[n], x1, y1, btnBtnColor,
btnTextColor, font, 1, 1, buttontexts[n] );

    // update the touch screen regions only during first build
    if (mode)
        x2 := gfx_Get(RIGHT_POS);
        y2 := gfx_Get(BOTTOM_POS);

        touchX1[n] := x1;
        touchY1[n] := y1;
        touchX2[n] := x2;
        touchY2[n] := y2;

        // calculate next button position
        x1 := x2 + xgap;
        if (x1 >= xorg + maxwidth)
            x1 := xorg;
            y1 := y2 + ygap;
        endif
    endif
endif
until (++n >= buttoncount);
endfunc

//=====
// do something with the key data
// In this example, we reconstitute the button name to a file name
// by appending ".4XE" and then call the file_Run command to
// run an application.
//=====
func sendkey()
    var p;

    p := buttontexts[keyval-1];
    to(tempstr); str_Printf(&p, "%s.4XE");

    txt_Set(TEXT_OPACITY, OPAQUE);
    txt_Set(FONT_ID, FONT_4);
    txt_MoveCursor(3, 0);

    print ("          ");

    if(file_Exists(str_Ptr(tempstr)))
        touch_Set(TOUCH_DISABLE); // disable the touch screen
        txt_Set(TEXT_COLOUR, ORANGE);
        print ("\rRUN: ", [STR] tempstr ); // run the required program
        pause(500);
        gfx_Cls();
        file_Run(str_Ptr(tempstr),0); // just run the prog, no args
    else
        txt_Set(TEXT_COLOUR, RED);
        print ("\rFAULT: ", [STR] tempstr ); // run required program
        pause(1000);
    endif
endfunc

//=====
// convert the touch co-ordinates to a key value
// returns 0 if no key down else return index 1..n of button
//=====
func readKeys(var x, var y)

```

```

var n, x1, y1, x2, y2, r;

n := 0;
r := 0;

while (n < buttoncount && !r)
  x1 := touchX1[n];
  y1 := touchY1[n];
  x2 := touchX2[n];
  y2 := touchY2[n];
  n++;
  if (x >= x1 && x < x2 && y >= y1 && y < y2) r := n;
wend

return r;
endfunc

//=====
func main()

var k, n, state, x, y;
var p, s, w, f;
redo:
w := 140;
f := FONT_4;
btnTextColor := BLACK;
btnBtnColor := LIGHTGREY;

gfx_Cls();
gfx_Set(BEVEL_WIDTH, 2);

txt_Set(FONT_ID, FONT_3);
print("Simple test for file_Run(...);\n");
print("Memory available = ", mem_Heap(), "\n");

if(!file_Mount())
  putstr("Disk not mounted");
  while(!file_Mount());
else
  putstr("Disk mounted\n");
endif

buttoncount := file_Count("*.4xe");
// count all the executable files on the drive
print("4XE File count = ", buttoncount, "\n");

n := buttoncount; // k holds entry count
if (!n)
  print("No 4XE executables\n");
// critical error, nothing to run!
repeat forever
endif

filenames := mem_AllocZ(n*13);
// allocate a buffer for the filenames
if(!filenames)
  print("Out of memory\n");
// critical error, could not allocate buffer
repeat forever
endif

to(filenames); file_Dir("*.4xe");
// load the filenames array

p := str_Ptr(filenames); // point to the string

//assign array of string pointers and truncate filename extensions
n := 0;

```



```

while ( n < buttoncount )
  buttontexts[n++] := p; // save pointer to the string
  p:=str_Find ( &p , "." ); // find end of required string
  str_PutByte(p++,'\0'); // change '.' to \0
  p := p + 4; // skip over "4XE\n"
wend

touch_Set(TOUCH_ENABLE); // enable the touch screen

redraw(buttoncount, f, 10, 80, w, 1);
// draw buttons for the first time

// now just stay in a loop
repeat
  state := touch_Get(TOUCH_STATUS); // get touchscreen status
  x := touch_Get(TOUCH_GETX);
  y := touch_Get(TOUCH_GETY);

  if(state == TOUCH_PRESSED) // if there's a press
    if (keyval := readKeys(x, y))
      vButtonState[keyval-1] := DOWN;
// put button in DOWN state
      redraw(buttoncount, f, 10, 80, w, 0);
// draw any button down states
    endif
  endif

  if(state == TOUCH_RELEASED)
// if there's a release
    if (keyval)
      vButtonState[keyval-1] := UP;
// restore the buttons UP state
      redraw(buttoncount, f, 10, 80, w, 0);
// draw any button up states
      sendkey();
// do something with the key data
      keyval := 0;
// because prog(main prog) gave up all its allocations for file_Exec,
// we have lost our file mount info and the directory list so we must
// re-establish these to be able to continue. A better approach to
// ensure total stability for the main program is to reset the system
      // with SystemReset()
      //=====
      // systemReset() // restart the main program
      // or
      goto redo; // re-mount disk, reload filenames
      //=====
    endif
  endif

  forever

  // mem_Free(filenamees);
  // no need to release buffer, this prog is in flash and never
  // exits.....
  // file_Unmount(); // ditto

endfunc
//=====

```

2.14.27. file_Exec(fname.4XE, arglistptr)

Syntax	file_Exec(fname.4XE, arglistptr);	
Arguments	fname.4XE, arglistptr	
	fname.4XE	name of the 4DGL child program to be loaded into RAM and executed.
	arglistptr	pointer to the list of arguments to pass to the new program or 0 if no arguments.
Returns	Value	
	Value	Returns the value from main in the called program.
Description	<p>This function is similar to file_Run, however, the main program in FLASH retains all memory allocations (eg file buffers, memory allocated with mem_Alloc etc)</p> <p>Returns like a function, current program calling program is kept active and control returns to it.</p> <p>If arglistptr is 0, no arguments are passed, else arglist points to an array, the first element being the number of elements in the array.</p> <p>func 'main' in the called program accepts the arguments.</p> <p>This function is similar to file_LoadFunction(...), however, the function argument list is passed by pointer, and the memory consumed by the function is released as soon as the function completes.</p>	
Example	<pre> Main Program: var args[4], l[50] ; func main() var i ; putstr("Mounting...\n"); // must mount uSD for file_Exec if (!(file_Mount())) while(!(file_Mount())) putstr("Drive not mounted..."); pause(200); gfx_Cls(); pause(200); wend endif for (i := 0; i < sizeof(l); i++) // init array that will be passed l[i] := i ; next args[0] := 2 ; // init arg count args[1] := 1234 ; // init arg 1, this cannot be changed args[2] := l ; // init arg 2 to address of l print("main Program\n") ; i := file_Exec("uSDProg.4fn", args) ; print("Back in main program\n") ; print("uSD Program returned ", i, "\n") ; // number from return statement for (i := 0; i < sizeof(l); i++) // find what changed in array </pre>	

```

        if (l[i] != i) print("l[" , i, "] was changed to " , l[i], "\n" )
;
    next
    print("Done") ;

    repeat
    forever

endfunc

Function on uSD:
func main(var j, var *l)          // parameters appear in the normal way
                                // The * shows that l will be indexed. It
                                // simply stops the compiler issuing a 'notice'
    txt_FGcolour(WHITE);
    print("In file_Exec's Program\n") ;
    print("Parms=", j, " ", l, "(ptr to l)\n") ; // can't change these
    print("Incrementing l[5] to ", ++l[5], "\n") ; // can change these
    print("Returning 188\n") ;           // can return a value
    txt_FGcolour(LIME);
    return 188 ;
endfunc

```

2.14.28. file_LoadImageControl(fname1, fname2, mode)

Syntax	file_LoadImageControl(fname1, fname2, mode);	
Arguments	fname1, fname2, mode	
	fname1	the control list filename "*.dat". Created from Graphics Composer.
	fname2	the image filename "*.gci". Created from Graphics Composer.
	mode	mode 0 : It is assumed that there is a graphics file with the file extension "fname2.gci". In this case, the images have been stored in a FAT16 file concurrently, and the

	<p>offsets that are derived from the "fname1.dat" file are saved in the image control so that the image control can open the file (*.gci) and use file_Seek(..) to get to the position of the image which can then automatically be displayed using file_Image(xpos, ypos, hSource).</p> <p>Mode 0 builds the image control quickly as it only scans the *.dat file for the file offsets and saves them in the relevant entries in the image control. The penalty is that images take longer to find when displayed due to file_Seek(..) overheads.</p> <p>mode 1 :</p> <p>It is assumed that there is a graphics file with the file extension "fname2.gci". In this case, the images have been stored in a FAT16 file concurrently, and the offset of the images are saved in the image control so that image file (*.gci) can be mapped to directly. The absolute cluster/sector is mapped so file seek does not need to be called internally. This means that there is no seek time penalty, however, the image list takes a lot longer to build, as all the seeking is done at control build time.</p> <p>Mode 2 :</p> <p>In this case, the images have been stored in a in a RAW partition of the uSD card, and the absolute address of the images are saved in the DAT file. This is the fastest operation of the image control as there is no seeking or other disk activity taking place.</p>												
Returns	Status												
	<p>Status</p> <p>Returns a handle (pointer to the memory allocation) to the image control list that has been created.</p> <p>Returns NULL if function fails.</p>												
Description	<p>Reads a control file to create an image list.</p> <p>When an image control is loaded, an array is built in ram. It consists of a 6 word header with the following entries as defined by the constants:</p> <table border="0"> <tr> <td>IMG_COUNT</td> <td>0</td> </tr> <tr> <td>IMG_ENTRYLEN</td> <td>1</td> </tr> <tr> <td>IMG_MODE</td> <td>2</td> </tr> <tr> <td>IMG_GCI_FILENAME</td> <td>3</td> </tr> <tr> <td>IMG_DAT_FILENAME</td> <td>4</td> </tr> <tr> <td>IMG_GCIFILE_HANDLE</td> <td>5</td> </tr> </table> <p>No images are stored in FLASH or RAM, the image control holds the index values for the absolute storage positions on the uSD card for RAW mode, or the cluster/sector position for formatted FAT16 mode.</p> <p>When an image control is no longer required, the memory can be released with:</p> <p>mem_Free(MyImageControlHandle);</p>	IMG_COUNT	0	IMG_ENTRYLEN	1	IMG_MODE	2	IMG_GCI_FILENAME	3	IMG_DAT_FILENAME	4	IMG_GCIFILE_HANDLE	5
IMG_COUNT	0												
IMG_ENTRYLEN	1												
IMG_MODE	2												
IMG_GCI_FILENAME	3												
IMG_DAT_FILENAME	4												
IMG_GCIFILE_HANDLE	5												

Example

```

#inherit "4DGL_16bitColours.fnc"

#constant OK 1
#constant FAIL 0

var p; // buffer pointer
var img; // handle for the image list
var n, exit, r;

//-----
// return true if screen touched, also sets ok flag
func CheckTouchExit()
    return (exit := (touch_Get(TOUCH_STATUS) == TOUCH_PRESSED)); //
if there's a press, exit
endfunc
//-----

func main()

    gfx_Cls();
    txt_Set(FONT_ID, FONT_2);
    txt_Set(TEXT_OPACITY, OPAQUE);

    touch_Set(TOUCH_ENABLE); // enable the touch screen

    print("heap=", mem_Heap(), " bytes\n"); // show the heap size

    r := OK; // return value
    exit := 0;

    if (!file_Mount())
        print("File error ", file_Error());
        while(!CheckTouchExit());
// just hang if we didnt get the image list
        r := FAIL;
        goto quit;
    endif

    print ("WAIT...building image list\n");

    // slow build, fast execution, higher memory requirement
    img := file_LoadImageControl("GFX2DEMO.dat", "GFX2DEMO.gci", 1);
    // build image control, returning a pointer to structure allocation

    if (img)
        print("image control=", [HEX] img, "\n");
// show the address of the image control allocation
    else
        putstr("Failed to build image control....\n");
        while(CheckTouchExit() == 0);
// just hang if we didnt get the image list
        r := FAIL;
        goto quit;
    endif

    print ("Loaded ", img[IMG_COUNT], " images\n");
    print ("\nTouch and hold to exit...\n");
    pause(2000);

    pause(3000);
    gfx_Cls();

    repeat
        n := 0;

        while(n < img[IMG_COUNT] && !exit) // go through all images

            CheckTouchExit(); // if there's a press, exit

```

```

        img_SetPosition( img, n, (ABS(RAND() % 240)), (ABS(RAND() %
320))); // spread out the images

        n++;

    wend

    img_Show(img, ALL); // update the entire control in 1 hit

until(exit);

quit:

    mem_Free(img); // release the image control

    file_Unmount(); // (program must release all resources)

    return r;

endfunc
//=====

```

2.14.29. file_Mount()

Syntax	file_Mount();	
Arguments	None	
Returns	Status	
	Status	Returns true if successful.
Description	Starts up the FAT16 disk file services and allocates a small 32 byte control block for subsequent	

use. When you open a file using `file_Open(..)`, a further $512 + 44 = 556$ bytes are attached to the FAT16 file control block. When you close a file using `file_Close(..)`, the 556 byte allocation is released leaving the 32 byte file control block. The `file_Mount()` function must be called before any other FAT16 file related functions can be used. The control block and all FAT16 file resources are completely released with `file_Unmount()`.

Example

```
if( !file_Mount() )
    repeat
        putstr("Disk not mounted");
        pause(200);
        gfx_Cls();
        pause(200);
    until( file_Mount() );
endif
```

2.14.30. file_Unmount()

Syntax	<code>file_Unmount();</code>
Arguments	None
Returns	None
Description	Release any buffers for FAT16 and unmount the Disk File System. This function is to be called to close the FAT16 file system.
Example	<code>file_Unmount(); // Unmount file system</code>

2.14.31. file_PlayWAV(fname)

Syntax	file_PlayWAV(fname);	
Arguments	fname	
	fname	Name of the wav file to be opened and played
Returns	value	
	value	<p>If there are no errors, returns number of blocks to play (1 to 32767)</p> <p>If errors occurred, the following is returned</p> <ul style="list-style-type: none"> -7 : Insufficient memory available for WAV buffer and file -6 : cant play this rate -5 : no data chunk found in first rsector -4 : no format data -3 : no wave chunk signature -2 : bad wave file format -1 : file not found
Description	<p>Open the wav file, decode the header to set the appropriate wave player parameters and set off the playing of the file as a background process.</p> <p>This function automatically grabs a chunk of memory for a file buffer, and a wave buffer. The minimum memory requirement is about 580 bytes for the disk io service and a minimum wave buffer size of 1024. The size of the wave buffer allocation can be increased by the snd_BufSize function.</p> <p>The default size 1024 bytes.</p> <p>Note: The memory is only required during the duration of play, and is automatically released while not in use.</p> <p>See “Sound Control Functions” for additional play control functions.</p>	
Example	<pre>print("\nding.wav\n"); for(n:=0; n<45; n++) pitch := NOTES[n]; print([UDEC] pitch, "\r"); snd_Pitch(pitch); file_PlayWAV("ding.wav"); while(snd_Playing()); //pause(500); next</pre>	

2.14.32. file_Rename(oldname, newname)

Syntax	<code>file_Rename(oldname, newname);</code>	
Arguments	oldname, newname	
	oldname	Name of the file to be renamed
	newname	Name of the file to be used as the new name
Returns	Status	
	Status	1: if successful 0: if unsuccessful
Description	<p>This function renames a file on the disk.</p> <p>Note: If the function fails, the appropriate error number is set in <code>file_Error()</code> if an invalid filename is specified, otherwise the cause will be a missing oldname or a pre-existing newname.</p>	
Example	<pre>res := file_Rename("myfile.txt", "myfile.bak") ;</pre>	

2.14.33. file_SetDate(handle, year, month, day, hour, minute, second)

Syntax	<code>file_SetDate(handle, year, month, day, hour, minute, second) ;</code>	
Arguments	handle, year, month, day, hour, minute, second	
	handle	The handle that references the file.
	year	The year the file was updated 1980-2099.
	month	The month the file was updated 1-12.
	day	The day the file was updated 1-31.
	hour	The hour the file was updated 0-23.
	minute	The minute the file was updated 0-59.
	Second	The second the file was updated 0-59.
Returns	Status	
	Status	1: if successful 0: if unsuccessful (Handle not valid, or Date/Time not valid)
Description	This function sets the modified date and time on an open file handle. The file must be closed at some future time for the date and time to be flushed to disk. Note that the FAT file system can only store even numbered seconds.	
Example	<code>ret := file_SetDate(hndl, 2014, 9, 15, 23, 58, 00);</code>	

2.15. Sound Control Functions

Summary of Functions in this section:

- `snd_Volume(var)`
- `snd_Pitch(pitch)`
- `snd_BufSize(var)`
- `snd_Stop()`
- `snd_Pause()`
- `snd_Continue()`
- `snd_Playing()`
- `snd_Freq()`

2.15.1. Snd_Volume(var)

Syntax	<code>Snd_Volume(var);</code>	
Arguments	var	
	var	sound playback volume
	The arguments can be a variable, array element, expression or constant	
Returns	None	
Description	Set the sound playback volume. Var must be in the range from 8 (min volume) to 127 (max volume). If var is less than 8, volume is set to 8, and if var > 127 it is set to 127.	
Example	<code>snd_Volume(127) ; // Set Volume to maximum</code>	

2.15.2. Snd_Pitch(pitch)

Syntax	Snd_Pitch(pitch);	
Arguments	pitch	
	pitch	Sample's playback rate. Minimum is 4KHz. Range is, 4000 – 65535.
	The arguments can be a variable, array element, expression or constant	
Returns	value	
	value	Returns sample's original sample rate.
Description	Sets the samples playback rate to a different frequency. Setting pitch to zero restores the original sample rate.	
Example	<code>snd_Pitch(7000); //Play the wav file with a sample frequency of 7KHz.</code>	

2.15.3. Snd_BufSize(var)

Syntax	<code>Snd_BufSize(var);</code>	
Arguments	var	
	var	<p>Specifies the buffer size.</p> <p>0 = 1024 bytes (default)</p> <p>1 = 2048 bytes</p> <p>2 = 4096 bytes</p> <p>3 = 8192 bytes</p>
	The arguments can be a variable, array element, expression or constant	
Returns	None.	
Description	<p>Specify the memory chunk size for the wavfile buffer, default size 1024 bytes. Depending on the sample size, memory constraints, and the sample quality, it may be beneficial to change the buffer size from the default size of 1024 bytes.</p> <p>This function is for control of a wav buffer, see the <code>file_PlayWAV(..)</code> ; function</p>	
Example	<code>snd_BufSize(1); // allocate a 2048 byte wav buffer</code>	

2.15.4. snd_Stop()

Syntax	<code>snd_Stop();</code>
Arguments	None
Returns	None
Description	Stop any sound that is currently playing, releasing buffers and closing any open wav file. This function is for control of a wav buffer, see the <code>file_PlayWAV(..)</code> ; function
Example	<code>snd_Stop(); // Stop, release buffers and close wav file</code>

2.15.5. snd_Pause()

Syntax	<code>snd_Pause();</code>
Arguments	None
Returns	None
Description	<p>Pause any sound that is currently playing, does nothing until sound is resumed with <code>snd_Continue()</code>.</p> <p>The sample can be terminated with <code>snd_Stop</code>.</p> <p>Buffers and closes any open wav file.</p> <p>This function is for control of a wav buffer, see the <code>file_PlayWAV(..)</code> ; function</p>
Example	<pre>snd_Pause(); // Pause Sound</pre>

2.15.6. snd_Continue()

Syntax	<code>snd_Continue();</code>
Arguments	None
Returns	None
Description	Resume any sound that is currently paused by <code>snd_Pause</code> . This function is for control of a wav buffer, see the <code>file_PlayWAV(..)</code> ; function
Example	<code>snd_Continue(); // Continue sound</code>

2.15.7. snd_Playing()

Syntax	snd_Playing();	
Arguments	None	
Returns	value	
	value	Number of 512 byte blocks to go.
Description	Returns 0 if sound has finished playing, else return number of 512 byte blocks to go. This function is for control of a wav buffer, see the file_PlayWAV(..) ; function	
Example	<code>count := snd_Playing(); // return number of sound blocks remaining</code>	

2.15.8. snd_Freq(frequency, duration)

Syntax	<code>snd_Freq(frequency, duration);</code>	
Arguments	<code>frequency, duration</code>	
	<code>frequency</code>	The frequency of the sound to produce, 10Hz is the minimum
	<code>duration</code>	The duration of the sound in milli seconds.
Returns	<code>status</code>	
	<code>status</code>	Returns TRUE if freq >= 10 and a wav file is not currently playing.
Description	Produces a pure square wave waveform on the audio output pin. This command is designed to drive Piezo transducers which require this sort of input. Whilst it also works on displays with a builtin amplifier the sound produced is extremely annoying.	
Example	<pre> snd_Freq(2731, 100); // produce a 100ms burst at the Piezo's resonant frequency. </pre>	

2.16. String Class Functions

Summary of Functions in this section:

- `str_Ptr(&var)`
- `str_GetD(&ptr, &var)`
- `str_GetW(&ptr, &var)`
- `str_GetHexW(&ptr, &var)`
- `str_GetC(&ptr, &var)`
- `str_GetByte(ptr)`
- `str_GetWord(ptr)`
- `str_PutByte(ptr, val)`
- `str_PutWord(ptr, val)`
- `str_Match(&ptr, *str)`
- `str_MatchI(&ptr, *str)`
- `str_Find(&ptr, *str)`
- `str_FindI(&ptr, *str)`
- `str_Length(ptr)`
- `str_Printf(&ptr, *format)`
- `str_Cat(&destination, &Source)`
- `str_CatN(&ptr, str, count)`
- `str_ByteMove(src, dest, count)`
- `str_Copy(dest, src)`
- `str_CopyN(dest, src, count)`

2.16.1. str_Ptr(&var)

Syntax	<code>str_Ptr(&var);</code>	
Arguments	var	
	var	Pointer to string buffer
Returns	Pointer	
	Pointer	Returned value is the byte pointer to string buffer.
Description	Return a byte pointer to a word region.	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer var n; var vars[3]; // for our results func main() to(buffer); print("0x1234 0b10011001 12345 abacus"); p := str_Ptr(buffer); //raise string pointer for the string functions while(str_GetW(&p, &vars[n++]) != 0); // read all the numbers till we //get a non number print(vars[0], "\n", vars[1], "\n", vars[2], "\n"); // print them out endfunc </pre>	

2.16.2. str_GetD(&ptr, &var)

Syntax	<code>str_GetD(&ptr, &var);</code>	
Arguments	&ptr, &var	
	ptr	Byte pointer to string.
	var	Destination for our result.
Returns	Status	
	Status	Returns TRUE if function succeeds, advancing ptr
Description	Convert number in a string to DWORD (myvar[2]). NB:- The address of the pointer must be passed so the function can advance it if required.	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer var n; var vars[6]; // for our results func main() to(buffer); print("100000 200000 98765432 abacus"); p := str_Ptr(buffer); // raise a string pointer so we can use the // string functions while(str_GetD(&p, &vars[n]) != 0) n:=n+2; //read all the numbers //till we get a non number print([HEX4] vars[1], ":" , [HEX4] vars[0], "\n"); // show the longs as hex numbers print([HEX4] vars[3], ":" , [HEX4] vars[2], "\n"); print([HEX4] vars[5], ":" , [HEX4] vars[4], "\n"); endfunc </pre>	

2.16.3. str_GetW(&ptr, &var)

Syntax	<code>str_GetW(&ptr, &var);</code>	
Arguments	&ptr, &var	
	ptr	Byte pointer to string.
	var	Destination for our result.
Returns	Status	
	Status	Returns TRUE if function succeeds, advancing ptr.
Description	Convert number in a string to WORD (myvar). NB:- The address of the pointer must be passed so the function can advance it if required.	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer var n; var vars[3]; // for our results func main() to(buffer); print("0x1234 0b10011001 12345 abacus"); p := str_Ptr(buffer); // raise a string pointer so we can use the // string functions while(str_GetW(&p, &vars[n++]) != 0); // read all the numbers till // we get a non number print(vars[0], "\n", vars[1], "\n", vars[2], "\n"); // print them out str_Printf (&p, "%s\n"); // numbers extracted, now just print // remainder of string endfunc </pre>	

2.16.4. str_GetHexW(&ptr, &var)

Syntax	<code>str_GetHexW(&ptr, &var);</code>	
Arguments	&ptr, &var	
	ptr	Byte pointer to string
	var	Destination for our result.
Returns	Status	
	Status	Returns TRUE if function succeeds, advancing ptr
Description	Convert hex number in a string to WORD (myvar). This function is for extracting 'raw' hex words with no "0x" prefix. Note: The address of the pointer must be passed so the function can advance it if required.	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer var n; var vars[4]; // for our results func main() to(buffer); print("1234 5678 9 ABCD"); p := str_Ptr(buffer); // raise a string pointer so we can use the // string functions while(str_GetHexW(&p, &vars[n++]) != 0); // read all the hex numbers // till we get a non number print(vars[0], "\n", vars[1], "\n" , vars[2], "\n", vars[3], "\n"); endfunc </pre>	

2.16.5. str_GetC(&ptr, &var)

Syntax	<code>str_GetC(&ptr, &var);</code>	
Arguments	&ptr, &var	
	ptr	Byte pointer to string.
	var	Destination for our result.
	The arguments can be a variable, array element, expression or constant	
Returns	Status	
	Status	Returns TRUE if function succeeds, advancing ptr.
Description	<p>Get next valid ascii char in a string to myvar. NB:- The address of the pointer must be passed so the function can advance it if required. The function returns 0 if end of string reached. Used for extracting single characters from a string.</p>	
Example	<pre> var p; // string pointer var n; var char; var buffer[100]; // 200 character buffer for a source string func main() to(buffer); print("Quick Brown Fox"); p := str_Ptr(buffer); // raise a string pointer so we can use the //string functions while(str_GetC(&p, &char)) print("p=",p," char is", [CHR] char); // print characters wend print("End of string"); endfunc </pre>	

2.16.6. str_GetByte(ptr)

Syntax	str_GetByte(ptr);	
Arguments	ptr	
	ptr	Address of byte array or string.
Returns	byte	
	byte	Returns the byte value at pointer location.
Description	Get a byte to myvar. Similar to "PEEKB" in basic. It is not necessary for byte pointer ptr to be word aligned	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var n, p; func main() to(buffer); print("Testing 1 2 3"); p := str_Ptr(buffer); // get a byte pointer from a word region n := 0; while (n <= str_Length(buffer)) print([HEX2] str_GetByte(p + n++), " "); // print all the chars hex // values wend endfunc </pre>	

2.16.7. str_GetWord(ptr)

Syntax	str_GetWord(ptr);	
Arguments	ptr	
	ptr	Byte pointer
Returns	Word	
	Word	Returns the word at pointer location.
Description	Get a word to myvar. Similar to PEEKW in basic. It is not necessary for byte pointer ptr to be word aligned	
Example	<pre> var p; // string pointer var buffer[10]; // array for 20 bytes func main() p := str_Ptr (buffer); // raise a string pointer str_PutWord (p+3, 100); // 'poke' the array str_PutWord (p+9, 200); str_PutWord (p+12, 400); print(str_GetWord(p + 3), "\n"); // 'peek' the array print(str_GetWord(p + 9), "\n"); print(str_GetWord(p + 12), "\n"); endfunc </pre>	

2.16.8. str_PutByte(ptr, val)

Syntax	<code>str_PutByte(ptr, val);</code>	
Arguments	<code>ptr, val</code>	
	ptr	Byte pointer to string
	val	Byte value to insert.
Returns	None	
Description	Put a byte value into a string buffer at ptr Similar to "POKEB" in basic It is not necessary for byte pointer ptr to be word aligned	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer func main() p := str_Ptr(buffer); // raise a string pointer so we can use the // string functions str_PutByte(p + 3, 'A'); // store some values str_PutByte(p + 4, 'B'); // store some values str_PutByte(p + 5, 'C'); // store some values str_PutByte(p + 7, 'D'); // store some values str_PutByte(p + 7, 0); // string terminator fprintf(vars[0], "\n", vars[1], "\n", vars[2], "\n"); // print them out p := p + 3; // offset to where we placed the chars fprintf(&p, "%s\n"); // print the result // nb, also, understand that the core print service // assumes a word aligned address so it starts at pos 4 // print([STR] &buffer[2]); endfunc </pre>	

2.16.9. str_PutWord(ptr, val)

Syntax	<code>str_PutWord(ptr, val);</code>	
Arguments	Ptr, val	
	ptr	Byte pointer
	val	Value to store.
Returns	None	
Description	Put a word value into a byte buffer at ptr, similar to "POKEW" in basic. It is not necessary for byte pointer ptr to be word aligned	
Example	<pre> var p; // string pointer var numbers[10]; // array for 20 bytes func main() p := str_Ptr (numbers); // raise a string pointer str_PutWord (p+3, 100); // 'poke' the array with some numbers str_PutWord (p+9, 200); str_PutWord (p+12, 400); print(str_GetWord(p + 3), "\n"); // 'peek' the array print(str_GetWord(p + 9), "\n"); print(str_GetWord(p + 12), "\n"); endfunc </pre>	

2.16.10. str_Match(&ptr, *str)

Syntax	<code>str_Match(&ptr, *str);</code>	
Arguments	<code>ptr, str</code>	
	<code>ptr</code>	Address of byte pointer to string buffer.
	<code>str</code>	Pointer string to match.
Returns	Value	
	Value	Returns 0 if no match, else advance ptr to the next position after the match and returns a pointer to the match position.
Description	<p>Case Sensitive match.</p> <p>Compares the string at position ptr in a string buffer to the string str, skipping over any leading spaces prior to the test. If a match occurs, ptr is advanced to the first position past the match, else ptr is not altered.</p> <p>Note: The address of the pointer must be passed so the function can advance it if required.</p>	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p, q; // string pointers var n; func main() to(buffer); print(" volts 240 "); // string to parse p := str_Ptr(buffer); // string pointer to be used // with string functions q := p; // match the start of the string with "volts" if (n := str_Match(&p, "volts")) str_Printf (&p, "%s\n"); // print remainder of string else print ("not found\n"); endif print ("startpos=" , q , "\nfindpos=" , n , "\nendpos=" , p); repeat forever endfunc </pre>	

2.16.11. str_MatchI(&ptr, *str)

Syntax	<code>str_MatchI(&ptr, *str);</code>	
Arguments	<code>ptr, str</code>	
	<code>ptr</code>	Address of byte pointer to string buffer.
	<code>str</code>	Pointer string to match.
Returns	Value	
	Value	Returns 0 if no match, else advance ptr to the next position after the match and returns a pointer to the match position.
Description	<p>Case Insensitive match.</p> <p>Compares the string at position ptr in a string buffer to the string str, skipping over any leading spaces prior to the test. If a match occurs, ptr is advanced to the first position past the match, else ptr is not altered.</p> <p>Note: The address of the pointer must be passed so the function can advance it if required.</p>	
Example	<pre>var buffer[100]; // 200 character buffer for a source string var p, q; // string pointers var n; func main() // string to parse to(buffer); print("The sun rises in the East"); p := str_Ptr(buffer); // string pointer to be used // with string functions q := p; // Will match if the string starts with "The", or "the" if (n := str_MatchI(&p, "the")) str_Printf (&p, "%s\n"); // print remainder of string else print ("not found\n"); endif print ("startpos=" , q , "\nfindpos=" , n , "\nendpos=" , p); repeat forever endfunc</pre>	

2.16.12. str_Find(&ptr, *str)

Syntax	<code>str_Find(&ptr, *str);</code>	
Arguments	<code>ptr, str</code>	
	<code>ptr</code>	Byte pointer to string buffer.
	<code>str</code>	String to find.
Returns	Value	
	Value	Returns 0 if not found. Returns the address of the first character of the match if successful.
Description	<p>Case Sensitive.</p> <p>Given the address of a pointer to a source string as the first argument, and a pointer to a test string as the second argument, attempts to find the position of the matching string in the source string. The test is performed with case sensitivity.</p> <p>NB:- The source pointer is not altered.</p>	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer var n; var strings[4]; // for our test strings func main() txt_Set (FONT_ID, FONT2); strings[0] := "useful" ; strings[1] := "string" ; strings[2] := "way" ; strings[3] := "class" ; to(buffer); print ("and by the way, the string class is rather useful "); // raise a string pointer so we can use the string functions p := str_Ptr(buffer); // offset into the buffer a little so we don't see word "way" p := p + 13; print("p=" , p , "\n\n"); // show the start point of our search n := 0; while (n < 4) print("\"", [STR] strings[n] , "\" is at pos " , str_Find(&p , strings[n++]) , "\n"); wend //note that p is unchanged print ("\nNOTE: p is unchanged, p=" , p); repeat forever endfunc </pre>	

2.16.13. str_FindI(&ptr, *str)

Syntax	<code>str_FindI(&ptr, *str);</code>	
Arguments	<code>ptr, str</code>	
	<code>ptr</code>	Byte pointer to string buffer.
	<code>str</code>	String to find.
Returns	Value	
	Value	Returns 0 if not found. Returns the address of the first character of the match if successful.
Description	<p>Case Insensitive.</p> <p>Given the address of a pointer to a source string as the first argument, and a pointer to a test string as the second argument, attempts to find the position of the matching string in the source string. The test is performed with case sensitivity, eg upper and lower case chars are accepted.</p> <p>NB:- The source pointer is not altered.</p>	
Example	<pre> var buffer[100]; // 200 character buffer for a source string var p; // string pointer var n; var strings[4]; // for our test strings func main() txt_Set (FONT_ID, FONT2); strings[0] := "USEFUL" ; strings[1] := "string" ; strings[2] := "way" ; strings[3] := "class" ; to(buffer); print ("and by the way, the String Class is rather useful "); // raise a string pointer so we can use the string functions p := str_Ptr(buffer); // offset into the buffer a little so we don't see word "way" p := p + 13; // show the start point of our search print("p=" , p , "\n\n"); n := 0; while (n < 4) print("\"" , [STR] strings[n] , "\" is at pos " , str_FindI (&p , strings[n++]) , "\n"); wend //note that p is unchanged print ("\nNOTE: p is unchanged, p=" , p); repeat forever endfunc </pre>	

2.16.14. str_Length(ptr)

Syntax	str_Length(ptr);	
Arguments	ptr	
	ptr	Pointer to string buffer.
Returns	Value	
	Value	Returns String length.
Description	Returns the length of a byte aligned string excluding terminator.	
Example	<pre> // Dynamic String Example func main() var a; var pa ; //This be a String pointer to a a := mem Alloc(200); // allocate a dynamic with undefined data mem_Set (a, 'X', 200); // fill it full of 'X's pa := str_Ptr(a); // raise a string pointer str_PutByte(pa+20,0); // Stick a string terminator in the array print ("a length:", str_Length(pa), "\n"); // show length of the // dynamic buffer // using the required string pointer mem_Free (a); // test is over, free up the memory repeat forever endfunc // Constant String Example func main() var b; b := "A string constant" ; // b is a pointer to a string constant print ("b length:", str_Length(b), "\n"); // show length of the // static string // a string constant is already a string pointer repeat forever endfunc // Array Example func main() var c[40]; // 80 character buffer for a source string var pc; // This will be a String pointer to c[] to (c); print ("An 'ASCIIIZ' string is terminated with a zero"); pc := str_Ptr(c); // raise a string pointer so we can use the // string functions print ("c length:", str_Length(pc), "\n"); // show length of the // 're-directed' string // using the required string pointer repeat forever endfunc </pre>	

2.16.15. str_Printf(&ptr, *format)

Syntax	<code>str_Printf(&ptr, *format);</code>	
Arguments	Ptr, format	
	ptr	Byte pointer to the input data (structure).
	format	<p>Format string.</p> <p>Note: The address of the pointer must be passed so the function can advance it as required.</p> <p>Note: The format specifier string can be a string pointer, allowing dynamic construction of the printing format.</p> <p>Format Specifiers:</p> <p>%c character</p> <p>%s string of characters</p> <p>%d signed decimal</p> <p>%ld long decimal</p> <p>%u unsigned decimal</p> <p>%lu long unsigned decimal</p> <p>%x hex byte</p> <p>%X hex word</p> <p>%lX hex long</p> <p>%b binary word</p> <p>%lb long binary word</p> <p>* indirection prefix (placed after '%' to specify indirect addressing)</p> <p>(number) width description (use between '%' and format specifier to set the field width).</p> <p>Note: If (number) is preceded by 0, the result is Left-pads with zeroes (0) instead of spaces.</p>
Returns	Pointer	
	Pointer	Returns the position of last extraction point. This is useful for processing by other string functions.
Description	This function prints a formatted string from elements derived from a structured byte region. There is only one input argument, the byte region pointer ptr which is automatically advanced as the format specifier string is processed. The format string is similar to the C language, however, there are a few differences, including the addition of the indirection token * (asterix).	
Example	<pre>var buffer[100]; // 200 character buffer for a source string var p, q; // string pointers var n; var m[20]; // for our structure example var format; // a pointer to a format string func main() var k;</pre>	

```
// string print example
to (buffer); print ( "\nHELLO WORLD" );

q := str_Ptr (buffer); // raise a string pointer so we can use the
                        // string functions
p := q;
str_Printf ( &p , "%8s" ); // only prints first 8 characters of
                        // string

putch ('\n');          // new line

p := q;
k := str_Printf ( &p , "%04s" ); // prints 4 leading spaces before
                        // string

putch ('\n'); // new line
print ( k );  // if required, the return value points to the last
              // source position and is returned for processing by
              // other string functions

// print structure elements example, make a demo structure

n := 0;
m[n++] := "Mrs Smith" ;
m[n++] := 200 ;
m[n++] := 300 ;
m[n++] := 0xAA55 ;
m[n++] := 500 ;

// make a demo format control string

format := "%*s\n%d\n%d\n%016b\n%04X" ; // format string for printing
                                         // structure m

// print the structure in the required format

p := str_Ptr (m);          // point to structure m
str_Printf (&p, format);   // use the format string to print the
                           // structure

endfunc
```

2.16.16. str_Cat(&destination, &source)

Syntax	<code>str_Cat(&destination, &source);</code>	
Arguments	destination, source	
	destination	Destination string address
	source	Source string address
Returns	Pointer	
	Pointer	Returns pointer to the destination.
Description	Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a new null-character is appended at the end of the new string formed by the concatenation of both in destination.	
Example	<pre> var buf[100]; // 200 character buffer for a source string func main() var p ; to(buf) ; print("Hello ") ; p := str_Ptr(buf) ; str_Cat(p,"There"); // Will append "There" to the end of buf print([STR] buf) ; repeat forever endfunc </pre>	

2.16.17. str_CatN(&ptr, str, count)

Syntax	<code>str_CatN(&ptr, str, count);</code>	
Arguments	ptr, str, count	
	ptr	Destination string address
	str	Source string address
	count	Number of characters to be concatenated.
Returns	Pointer	
	Pointer	Returns pointer to the destination.
Description	The number of characters copied is limited by "count". The terminating null character in destination is overwritten by the first character of source, and a new null-character is appended at the end of the new string formed by the concatenation of both in destination.	
Example	<pre> var buf[100]; // 200 character buffer for a source string func main() var p ; to(buf) ; print("Sun ") ; p := str_Ptr(buf) ; str_CatN(p, "Monday", 3); // Concatenate "Mon" to the end of buf print([STR] buf) ; repeat forever endfunc </pre>	

2.16.18. str_ByteMove(src, dest, count)

Syntax	<code>str_ByteMove(src, dest, count);</code>	
Arguments	<code>src, dest, count</code>	
	src	Points to byte aligned source.
	dest	Points to byte aligned destination.
	count	Number of bytes to transfer.
Returns	Pointer	
	Pointer	Returns a pointer to the end of the destination (which is "dest" + "count").
Description	Copy bytes from "src" to "dest", stopping only when "count" is exhausted. No terminator is appended, it is purely a byte copy, and any zeroes encountered will also be copied.	
Example	<pre> var src, dest, mybuf1[10], mybuf2[10]; // string pointers and two 20 byte buffers to(mybuf1); putstr("TESTING 123"); src := strPtr(mybuf1); dest := str_Ptr(mybuf2); src += 6; // move src pointer to "G 123" str_ByteMove(src, dest, 6); // move to second buffer (including the zero terminator) putstr(mybuf2); // print result nextpos := str_ByteMove(s, d, 100); </pre>	

2.16.19. str_Copy(dest, src)

Syntax	<code>str_Copy(dest, src);</code>	
Arguments	<code>dest, src</code>	
	dest	Points to byte aligned destination.
	src	Points to byte aligned source.
Returns	Pointer	
	Pointer	Returns a pointer to the 0x00 string terminator at the end of "dest" (which is "dest" + str_Length(src);).
Description	Copy a string from "src" to "dest", stopping only when the end of source string "src" is encountered (0x00 terminator). The terminator is always appended, even if "src" is an empty string.	
Example	<code>nextplace := str_Copy(d, s);</code>	

2.16.20. str_CopyN(dest, src, count)

Syntax	<code>str_CopyN(dest, src, count);</code>	
Arguments	<code>dest, src, count</code>	
	dest	Points to byte aligned destination.
	src	Points to byte aligned source.
	count	Maximum number of bytes to copy.
Returns	Pointer	
	Pointer	Returns a pointer to the 0x00 string terminator at the end of "dest" (which is "dest" + str_Length(src);).
Description	Copy a string from "src" to "dest", stopping only when "count" is exhausted, or end of source string "str" is encountered (0x00 string terminator). The terminator is always appended, even if "count" is zero, or "src" is a null string.	
Example	<pre>nextplace := str_CopyN(d, s, 100);</pre>	

2.17. Touch Screen Functions

Summary of Functions in this section:

- touch_DetectRegion(x1, y1, x2, y2)
- touch_Set(mode)
- touch_Get(mode)
- touch_TestArea(&rect)
- touch_TestBox(&rect)

2.17.1. touch_DetectRegion(x1, y1, x2, y2)

Syntax	<code>touch_DetectRegion(x1, y1, x2, y2);</code>	
Arguments	X1, y1, x2, y2	
	x1	specifies the horizontal position of the top left corner of the region.
	y1	specifies the vertical position of the top left corner of the region.
	x2	specifies the horizontal position of the bottom right corner of the region.
	y2	specifies the vertical position of the bottom right corner of the region.
Returns	None	
Description	Specifies a new touch detect region on the screen. This setting will filter out any touch activity outside the region and only touch activity within that region will be reported by the status poll <code>touch_Get(TOUCH_STATUS)</code> function.	

2.17.2. touch_Set(mode)

Syntax	<code>touch_Set(mode);</code>	
Arguments	mode	
	mode	<p>mode = TOUCH_ENABLE (Mode 0) Enable Touch Screen</p> <p><code>touch_Set(TOUCH_ENABLE);</code> Enables and initialises Touch Screen hardware</p> <p>mode = TOUCH_DISABLE (Mode 1) Disable Touch Screen</p> <p><code>touch_Set(TOUCH_DISABLE);</code> Disables the Touch Screen. Note: Touch Screen task runs in the background and disabling it when not in use will free up extra resources for 4DGL CPU cycles.</p> <p>mode = TOUCH_REGIONDEFAULT (Mode 2) Default Touch Region</p> <p><code>touch_Set(TOUCH_REGIONDEFAULT);</code> This will reset the current active region to default which is the full screen area</p>
Returns	None	
Description	Sets various Sets various Touch Screen related parameters.	
Example	<code>touch_Set(TOUCH_ENABLE); // .</code>	

2.17.3. touch_Get(mode)

Syntax	touch_Get(mode);	
Arguments	mode	
	mode	mode = TOUCH_STATUS (Mode 0): Get Status mode = TOUCH_GETX (Mode 1): Get X coordinates mode = TOUCH_GETY (Mode 2): Get Y coordinates
Returns	Value	mode = TOUCH_STATUS (Mode 0) Returns the various states of the touch screen 0 = NOTOUCH 1 = TOUCH_PRESSED 2 = TOUCH_RELEASED 3 = TOUCH_MOVING mode = TOUCH_GETX (Mode 1) Returns the X coordinates of the touch reported by mode 0 mode = TOUCH_GETY (Mode 2) Returns the Y coordinates of the touch reported by mode 0
Description	Returns various Touch Screen parameters to caller.	
Example	<pre> state := touch_Get(TOUCH_STATUS); // get touchscreen status x := touch_Get(TOUCH_GETX); y := touch_Get(TOUCH_GETY); if (state == TOUCH_PRESSED) // see if Exit hit if (x > 170 && y > 280) // EXIT button gfx_Cls(); exit := -1; endif if (vertical) if (x > 170 && (y > 240 && y < 270))// Horiz button vertical := 0; exit := 1; endif else if (x > 170 && (y > 200 && y < 230))// Vert button vertical := 1; exit := 2; endif endif endif </pre>	

2.17.4. touch_TestArea(&rect)

Syntax	<code>touch_TestArea(&rect);</code>	
Arguments	<code>rect</code>	
	<code>rect</code>	An array of 4 vars, x1, y1, x2, y2 (using absolute co-ordinates)
Returns	<code>Status</code>	
	<code>Status</code>	Returns TRUE if last touch co-ordinates are within the absolute co-ordinate test area.
Description	The touch_TestArea function creates a test area based on the parameters in <code>rect</code> , and returns true if the last touch resided within the test area. <code>rect</code> is an array of 4 vars, x1, y1, x2, y2 (using absolute co-ordinates)	
Example	<pre>var r[4]; r[0]:=100; r[1]:=200; r[2]:=119; r[3]:=129; result := touch_Test(r); // check area at 100,200,119,229</pre>	

2.17.5. touch_TestBox(&rect)

Syntax	<code>touch_TestBox(&rect);</code>	
Arguments	rect	
	rect	An array of 4 vars, x1, y1, width, height
Returns	Status	
	Status	Returns TRUE if last touch co-ordinates are within the boxed test area.
Description	<p>The touch_TestArea function creates a test box based on the parameters in rect, and returns true if the last touch resided within the boxed test area.</p> <p>rect is an array of 4 vars, x1, y1, width, height (using boxed co-ordinates)</p>	
Example	<pre>var r[4]; r[0]:=100; r[1]:=200; r[2]:=20; r[3]:=30; result := touch_Test(r); // check area at 100,200,119,229</pre>	

2.18. Image Control Functions

Summary of Functions in this section:

- `img_SetPosition(handle, index, xpos, ypos)`
- `img_Enable(handle, index)`
- `img_Disable(handle, index)`
- `img_Darken(handle, index)`
- `img_Lighten(handle, index)`
- `img_SetWord(handle, index, offset, word)`
- `img_GetWord(handle, index, offset)`
- `img_Show(handle, index)`
- `img_SetAttributes(handle, index, value)`
- `img_ClearAttributes(handle, index, value)`
- `img_Touched(handle, index)`
- `img_SelectReadPosition(handle, index, frame, x, y)`
- `img_SequentialRead(count, ptr)`

2.18.1. `img_SetPosition(handle, index, xpos, ypos)`

Syntax	<code>img_SetPosition(handle, index, xpos, ypos);</code>	
Arguments	handle, index, xpos, ypos	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
	xpos	Top left horizontal screen position where image is to be displayed.
	ypos	Top left vertical screen position where image is to be displayed.
Returns	Status	
	Status	Returns TRUE if index OK and function successful
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Sets the position where the image will next be displayed. Returns TRUE if index was ok and function was successful. (the return value is usually ignored).</p> <p>You may turn off an image so when <code>img_Show()</code> is called, the image will not be shown.</p> <p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p>	
Example	<pre>// make a simple 'window' gfx_Panel(PANEL_RAISED, 0, 0, 239, 239, GRAY); img_SetPosition(Ihndl, BTN_EXIT, 224,2); //set checkout box position img_Enable(Ihndl, BTN_EXIT); //enable checkout box</pre>	

2.18.2. `img_Enable(handle, index)`

Syntax	<code>img_Enable(handle, index);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
Returns	Status	
	Status	Returns TRUE if index OK and function successful
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Enables a selected image in the image list. Returns TRUE if index was ok and function was successful. This is the default state so when <code>img_Show()</code> is called all the images in the list will be shown.</p> <p>To enable all of the images in the list at the same time set index to -1.</p> <p>To enable a selected image, use the image index number.</p>	
Example	<code>r := img_Enable(hImageList, imagenum);</code>	

2.18.3. `img_Disable(handle, index)`

Syntax	<code>img_Disable(handle, index);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
Returns	Status	
	Status	Returns TRUE if index OK and function successful
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Disables an image in the image list. Returns TRUE if index was ok and function was successful. Use this function to turn off an image so that when <code>img_Show()</code> is called the selected image in the list will not be shown.</p> <p>To disable all of the images in the list at the same time set index to -1.</p>	
Example	<code>r := img_Disable(hImageList, imagenum);</code>	

2.18.4. `img_Darken(handle, index)`

Syntax	<code>img_Darken(handle, index);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
Returns	Status	
	Status	Returns TRUE if index OK and function successful
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Darken an image in the image list. Returns TRUE if index was ok and function was successful. Use this function to darken an image so that when <code>img_Show()</code> is called the control will take effect. To darken all of the images in the list at the same time set index to -1.</p> <p>Note: This feature will take effect one time only and when <code>img_Show()</code> is called again the darkened image will revert back to normal.</p>	
Example	<code>r := img_Darken(hImageList, imagenum);</code>	

2.18.5. `img_Lighten(handle, index)`

Syntax	<code>img_Lighten(handle, index);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
Returns	Status	
	Status	Returns TRUE if index OK and function successful
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Lighten an image in the image list. Returns TRUE if index was ok and function was successful. Use this function to lighten an image so that when <code>img_Show()</code> is called the control will take effect. To lighten all of the images in the list at the same time set index to -1.</p> <p>Note: This feature will take effect one time only and when <code>img_Show()</code> is called again the lightened image will revert back to normal.</p>	
Example	<code>r := img_Lighten(hImageList, imagenum);</code>	

2.18.6. `img_SetWord(handle, index, offset, word)`

Syntax	<code>img_SetWord(handle, index, offset, word);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
	offset	Offset of the required word in the image entry
	word	The word to be written to the entry
Returns	Status	
	Status	TRUE if successful, usually ignored
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Set specified word in an image entry. Returns TRUE if successful, return value usually ignored.</p> <pre> IMAGE_XPOS 2 // WORD image location X IMAGE_YPOS 3 // WORD image location Y IMAGE_FLAGS 6 // WORD image flags IMAGE_DELAY 7 // WORD inter frame delay IMAGE_INDEX 9 // WORD current frame IMAGE_TAG 12 // WORD user variable #1 IMAGE_TAG2 13 // WORD user variable #2 </pre> <p>Note: Not all Constants are listed as some are Read Only.</p> <p><code>img_Show(..)</code> will now show error box for out of range video frames. Also, if frame is set to -1, just a rectangle will be drawn in background colour to blank an image. It applies to PmmC R29 or above.</p>	
Example	<pre> func cat() var private frame := 0; // start with frame 0 var private image := SPRITE_CAT; // cat image, can be changed with // cat.image := xxx var private speed := 30; img_SetWord(Ihndl, image, IMAGE_INDEX, frame++); frame := frame % img_GetWord(Ihndl, image, IMAGE_FRAMES); img_Show(Ihndl, image); sys_SetTimer(TIMER3, speed); // reset the event timer endfunc </pre>	

2.18.7. `img_GetWord(handle, index, offset)`

Syntax	<code>img_GetWord(handle, index, offset);</code>																																											
Arguments	handle, index																																											
	handle	Pointer to the Image List.																																										
	index	Index of the images in the list.																																										
	offset	Offset of the required word in the image entry																																										
Returns	Value																																											
	value	Returns the image entry in the list.																																										
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Returns specified word from an image entry.</p> <table> <tr><td>IMAGE_LOWORD</td><td>0</td><td>// WORD image address LO</td></tr> <tr><td>IMAGE_HIWORD</td><td>1</td><td>// WORD image address HI</td></tr> <tr><td>IMAGE_XPOS</td><td>2</td><td>// WORD image location X</td></tr> <tr><td>IMAGE_YPOS</td><td>3</td><td>// WORD image location Y</td></tr> <tr><td>IMAGE_WIDTH</td><td>4</td><td>// WORD image width</td></tr> <tr><td>IMAGE_HEIGHT</td><td>5</td><td>// WORD image height</td></tr> <tr><td>IMAGE_FLAGS</td><td>6</td><td>// WORD image flags</td></tr> <tr><td>IMAGE_DELAY</td><td>7</td><td>// WORD inter frame delay</td></tr> <tr><td>IMAGE_FRAMES</td><td>8</td><td>// WORD number of frames</td></tr> <tr><td>IMAGE_INDEX</td><td>9</td><td>// WORD current frame</td></tr> <tr><td>IMAGE_CLUSTER</td><td>10</td><td>// WORD image start cluster pos (for FAT16 only)</td></tr> <tr><td>IMAGE_SECTOR</td><td>11</td><td>// WORD image start sector in cluster pos (for FAT16 only)</td></tr> <tr><td>IMAGE_TAG</td><td>12</td><td>// WORD user variable #1</td></tr> <tr><td>IMAGE_TAG2</td><td>13</td><td>// WORD user variable #2</td></tr> </table>		IMAGE_LOWORD	0	// WORD image address LO	IMAGE_HIWORD	1	// WORD image address HI	IMAGE_XPOS	2	// WORD image location X	IMAGE_YPOS	3	// WORD image location Y	IMAGE_WIDTH	4	// WORD image width	IMAGE_HEIGHT	5	// WORD image height	IMAGE_FLAGS	6	// WORD image flags	IMAGE_DELAY	7	// WORD inter frame delay	IMAGE_FRAMES	8	// WORD number of frames	IMAGE_INDEX	9	// WORD current frame	IMAGE_CLUSTER	10	// WORD image start cluster pos (for FAT16 only)	IMAGE_SECTOR	11	// WORD image start sector in cluster pos (for FAT16 only)	IMAGE_TAG	12	// WORD user variable #1	IMAGE_TAG2	13	// WORD user variable #2
IMAGE_LOWORD	0	// WORD image address LO																																										
IMAGE_HIWORD	1	// WORD image address HI																																										
IMAGE_XPOS	2	// WORD image location X																																										
IMAGE_YPOS	3	// WORD image location Y																																										
IMAGE_WIDTH	4	// WORD image width																																										
IMAGE_HEIGHT	5	// WORD image height																																										
IMAGE_FLAGS	6	// WORD image flags																																										
IMAGE_DELAY	7	// WORD inter frame delay																																										
IMAGE_FRAMES	8	// WORD number of frames																																										
IMAGE_INDEX	9	// WORD current frame																																										
IMAGE_CLUSTER	10	// WORD image start cluster pos (for FAT16 only)																																										
IMAGE_SECTOR	11	// WORD image start sector in cluster pos (for FAT16 only)																																										
IMAGE_TAG	12	// WORD user variable #1																																										
IMAGE_TAG2	13	// WORD user variable #2																																										
Example	<code>myvar := img_GetWord(hndl, 5, IMAGE_YPOS); //</code>																																											

2.18.8. `img_Show(handle, index)`

Syntax	<code>img_Show(handle, index);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
Returns	Status	
	Status	Returns TRUE if successful, usually ignored
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Enable the displaying of the image entry in the image control. Returns TRUE if successful but return value is usually ignored.</p>	
Example	<code>img_Show(hImageList, imagenum);</code>	

2.18.9. `img_SetAttributes(handle, index, value)`

Syntax	<code>img_SetAttributes(handle, index, value);</code>	
Arguments	handle, index, value	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
	value	Refers to various bits in the image control entry (see image attribute flags)
Returns	Status	
	Status	Returns TRUE if successful, usually ignored
Description	<p>This function SETS one or more bits in the IMAGE_FLAGS field of an image control entry. "value" refers to various bits in the image control entry (see image attribute flags). A '1' bit in the "value" field SETS the respective bit in the IMAGE_FLAGS field of the image control entry.</p> <pre> I_ENABLED 0x8000 // bit 15, set for image enabled I_DARKEN 0x4000 // bit 14, display dimmed I_LIGHTEN 0x2000 // bit 13, display bright I_TOUCHED 0x1000 // bit 12, touch test result I_Y_LOCK 0x0800 // bit 11, stop Y movement I_X_LOCK 0x0400 // bit 10, stop X movement I_TOPMOST 0x0200 // bit 9, draw on top of other images next update I_STAYONTOP 0x0100 // bit 8, draw on top of other images always </pre>	
Example	<pre> : : img_Enable(Ihndl, SPRITE_CAT); // we'll also use small cat video img_SetAttributes(Ihndl, SPRITE_CAT, I_NOGROUP); img_SetPosition(Ihndl, SPRITE_CAT, 160, 180); // set its position : </pre>	

2.18.10. `img_ClearAttributes(handle, index, value)`

Syntax	<code>img_ClearAttributes(handle, index, value);</code>																									
Arguments	handle, index, value																									
	handle	Pointer to the Image List.																								
	index	Index of the images in the list.																								
	value	a '1' bit indicates that a bit should be set and a '0' bit indicates that a bit is not altered. Note: if index is set to -1, the attribute is altered in ALL of the entries in the image list . The constant ALL is set to -1 specifically for this purpose.																								
Returns	Status																									
	Status	Returns TRUE if successful, usually ignored																								
Description	<p>Clear various image attribute flags in a image control entry. (see image attribute flags below)</p> <p>Image attribute flags may be combined with the + or operators, eg:- <code>img_ClearAttributes(hndl, ALL, I_Y_LOCK I_X_LOCK); // allow all images to move in any direction</code></p> <p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Image attribute flags</p> <table> <tr> <td><code>I_ENABLED</code></td> <td><code>0x8000</code></td> <td>// bit 15, set for image enabled</td> </tr> <tr> <td><code>I_DARKEN</code></td> <td><code>0x4000</code></td> <td>// bit 14, display dimmed</td> </tr> <tr> <td><code>I_LIGHTEN</code></td> <td><code>0x2000</code></td> <td>// bit 13, display bright</td> </tr> <tr> <td><code>I_TOUCHED</code></td> <td><code>0x1000</code></td> <td>// bit 12, touch test result</td> </tr> <tr> <td><code>I_Y_LOCK</code></td> <td><code>0x0800</code></td> <td>// bit 11, stop Y movement</td> </tr> <tr> <td><code>I_X_LOCK</code></td> <td><code>0x0400</code></td> <td>// bit 10, stop X movement</td> </tr> <tr> <td><code>I_TOPMOST</code></td> <td><code>0x0200</code></td> <td>// bit 9, draw on top of other images next update</td> </tr> <tr> <td><code>I_STAYONTOP</code></td> <td><code>0x0100</code></td> <td>// bit 8, draw on top of other images always</td> </tr> </table>		<code>I_ENABLED</code>	<code>0x8000</code>	// bit 15, set for image enabled	<code>I_DARKEN</code>	<code>0x4000</code>	// bit 14, display dimmed	<code>I_LIGHTEN</code>	<code>0x2000</code>	// bit 13, display bright	<code>I_TOUCHED</code>	<code>0x1000</code>	// bit 12, touch test result	<code>I_Y_LOCK</code>	<code>0x0800</code>	// bit 11, stop Y movement	<code>I_X_LOCK</code>	<code>0x0400</code>	// bit 10, stop X movement	<code>I_TOPMOST</code>	<code>0x0200</code>	// bit 9, draw on top of other images next update	<code>I_STAYONTOP</code>	<code>0x0100</code>	// bit 8, draw on top of other images always
<code>I_ENABLED</code>	<code>0x8000</code>	// bit 15, set for image enabled																								
<code>I_DARKEN</code>	<code>0x4000</code>	// bit 14, display dimmed																								
<code>I_LIGHTEN</code>	<code>0x2000</code>	// bit 13, display bright																								
<code>I_TOUCHED</code>	<code>0x1000</code>	// bit 12, touch test result																								
<code>I_Y_LOCK</code>	<code>0x0800</code>	// bit 11, stop Y movement																								
<code>I_X_LOCK</code>	<code>0x0400</code>	// bit 10, stop X movement																								
<code>I_TOPMOST</code>	<code>0x0200</code>	// bit 9, draw on top of other images next update																								
<code>I_STAYONTOP</code>	<code>0x0100</code>	// bit 8, draw on top of other images always																								
Example	<code>img_ClearAttributes(hndl, 5, value); //</code>																									

2.18.11. img_Touched(handle, index)

Syntax	<code>img_Touched(handle, index);</code>	
Arguments	handle, index	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
Returns	Status	
	Status	Returns index if Touched Returns -1 if not Touched
Description	<p>This function requires that an image control has been created with the <code>file_LoadImageControl(...);</code> function.</p> <p>Returns index if image touched or returns -1 image not touched. If index is passed as -1 the function tests all images and returns -1 if image not touched or returns index.</p>	
Example	<pre> if(state == TOUCH_PRESSED) n := img_Touched(Ihndl, -1); //scan image list, looking for a touch if(n != -1) last := n; button := n; img_Lighten(Ihndl, n); //lighten the button touched img_Show(Ihndl, -1); // restore the images endif endif </pre>	

2.18.12. img_SelectReadPosition(handle, index, frame, xpos, ypos)

Syntax	img_SelectReadPosition(handle, index, frame, xpos, ypos);	
Arguments	handle, index, frame, xpos, ypos	
	handle	Pointer to the Image List.
	index	Index of the images in the list.
	frame	Frame to read if the 'image' is a video, else 0
	xpos	Image location, x position (top left corner)
	ypos	Image location, Y position (top left corner)
Returns	Status	
	Status	Returns TRUE if index was ok and function successful
Description	<p>This Functions sets a position in an image control for sequential reading of pixels from the uSD card (fat16 or raw modes supported)</p> <p>No image window area is set, the image will not be shown</p> <p>This function provides a means of preparing to load an image, or part of an image, to an array. (see img_SequentialRead)</p>	
Example	<pre> var subpic[55*60]; func main() var i, h, p, w ; if (!file_Mount()) putstr("\nDrive not mounted..."); // simplistic error handling repeat forever endif handle := file_LoadImageControl("Nemo240.dat", "Nemo240.gci", 1); h := img_GetWord(handle, 0, IMAGE_HEIGHT); w := img_GetWord(handle, 0, IMAGE_WIDTH); img_SelectReadPosition(handle, 0, 520, 55, 63); p := subpic ; for (i := 0; i < 60; i++) img_SequentialRead(55, p); // read pixels from selected read position of an image p += 55 ; img_SequentialRead(w-55, 0); // skip to next line next gfx_WriteGRAMarea(0, 240, 54, 299, subpic); img_SetWord(handle, 0, IMAGE_INDEX, 520); // frame is 0 to 604 img_Show(handle,0); repeat forever // intial testing only endfunc </pre>	

2.18.13. `img_SequentialRead(count, ptr)`

Syntax	<code>img_SequentialRead(count, ptr);</code>	
Arguments	<code>count, ptr</code>	
	<code>count</code>	Number of Pixels to read
	<code>ptr</code>	A pointer to an array to read count pixels into
Returns	<code>Status</code>	
	<code>Status</code>	Returns TRUE if index was ok and function successful
Description	<p>Once a position has ben set with the <code>img_SelectReadPosition</code> function, this function can then used for sequential reading of pixels from image storage.</p> <p>If "<code>ptr</code>" is 0, "<code>count</code>" pixels from the stream are simply skipped</p> <p>If "<code>ptr</code>" is 1, "<code>count</code>" pixels are written to the GRAM area</p> <p>"<code>ptr</code>" must point to a valid array that is at least the size of "<code>count</code>", or part of an image, to an array. (see <code>img_SelectReadPosition</code>)</p>	
Example	See <code>img_SelectReadPosition</code> example	

2.19. Memory Allocation Functions

Summary of Functions in this section:

- mem_Alloc(size)
- mem_Allocv(size)
- mem_Allocz(size)
- mem_Realloc(ptr, size)
- mem_Free(allocation)
- mem_Heap()
- mem_Set(ptr, char, size)
- mem_Copy(source, destination, count)
- mem_Compare(ptr1, ptr2, count)
- mem_ArrayOp1(memarray, count, op, value)
- mem_ArrayOP2(memarray1, memarray2, count, op, value)

2.19.1. mem_Alloc(size)

Syntax	<code>mem_Alloc(size);</code>	
Arguments	size (byte)	
	size	Specifies the number of bytes that's allocated from the heap.
Returns	value	
	value	Returned value is the pointer (Word) to the allocation if successful. If function fails returns a null (0).
Description	Allocate a block of memory to pointer myvar. The allocated memory contains garbage but is a fast allocation. The block must later be released with <code>mem_Free(myvar);</code>	
Example	<code>myvar := mem_Alloc(100);</code>	

2.19.2. mem_AllocV(size)

Syntax	<code>mem_AllocV(size);</code>	
Arguments	size (Byte)	
	size	Specifies the number of bytes that's allocated from the heap.
Returns	Value	
	Value	Returned value is the pointer (Word) to the allocation if successful. If function fails returns a null (0).
Description	Allocate a block of memory to pointer myvar. The block of memory is filled with initial signature values. The block starts with A5,5A then fills with incrementing number eg:- A5,5A,00,01,02,03...FF,00,11.... This can be helpful when debugging. The block must later be released with mem_Free(myvar).	
Example	<code>myvar := mem_AllocV(100);</code>	

2.19.3. mem_Allocz(size)

Syntax	<code>mem_Allocz(size);</code>	
Arguments	size	
	size	Specifies the number of bytes that's allocated from the heap.
Returns	Value	
	Value	Returned value is the pointer to the allocation if successful. If function fails returns a null (0).
Description	Allocate a block of memory to pointer myvar. The block of memory is filled with zeros. The block must later be released with <code>mem_Free(myvar);</code>	
Example	<code>myvar := mem_Allocz(100);//</code>	

2.19.4. mem_Realloc(&ptr, size)

Syntax	<code>mem_Realloc(&ptr, size);</code>	
Arguments	<code>ptr, size</code>	
	ptr	Specifies the new location to reallocate the memory block.
	size	Specifies the number of bytes of the block.
Returns	Status	
	Status	See the Description.
Description	The function may move the memory block to a new location, in which case the new location is returned. The content of the memory block is preserved up to the lesser of the new and old sizes, even if the block is moved. If the new size is larger, the value of the newly allocated portion is indeterminate. In case that ptr is NULL, the function behaves exactly as mem_Alloc(), assigning a new block of size bytes and returning a pointer to the beginning of it. In case that the size is 0, the memory previously allocated in ptr is deallocated as if a call to mem_Free(myvar) was made, and a NULL pointer is returned.	
Example	<code>myvar := mem_Realloc(myptr, 100); //</code>	

2.19.5. mem_Free(allocation)

Syntax	<code>mem_Free(allocation);</code>	
Arguments	allocation	
	allocation	specifies the location of memory block to free up.
Returns	Status	
	Status	Returns non-zero if function is successful Returns 0 if the function fails.
Description	The function de-allocates a block of memory previously created with <code>mem_Alloc(...)</code> , <code>mem_AllocV(...)</code> or <code>mem_AllocZ(...)</code> .	
Example	<code>test := mem_Free(myvar); //</code>	

2.19.6. mem_Heap()

Syntax	<code>mem_Heap();</code>	
Arguments	None	
Returns	Value	
	Value	Returns the largest available byte memory chunk in the heap.
Description	Returns byte size of the largest chunk of memory available in the heap.	
Example	<code>howmuch := mem_Heap();</code>	

2.19.7. mem_Set(ptr, char, size)

Syntax	mem_Set(ptr, char, size);	
Arguments	ptr, char, size	
	ptr	Specifies the memory block.
	char	Specifies the value to fill the block with.
	size	Specifies the size of the block in Bytes.
Returns	Pointer	
	Pointer	Returns the pointer.
Description	Fill a block of memory with a byte value.	
Example	<pre> var mybuf[5]; var i; func main() mem_Set(mybuf,0x55,5); //Only fills half of mybuf[] for(i:=0;i<sizeof(mybuf);i++) //Show what is in the buffer print(" 0x",[HEX]mybuf[i]); next mem_Set(mybuf,0xAA,sizeof(mybuf)*2); //Fill entire buffer print("\n"); //New line for(i:=0;i<sizeof(mybuf);i++) print(" 0x",[HEX]mybuf[i]); next repeat forever </pre>	

2.19.8. mem_Copy(source, destination, count)

Syntax	<code>mem_Copy(source, destination, count);</code>	
Arguments	source, destination, count	
	source	Specifies the source memory block.
	destination	Specifies the destination memory block.
	count	Specifies the size of the blocks in bytes.
Returns	Pointer	
	Pointer	Returns source.
Description	<p>Copy a word aligned block of memory from source to destination.</p> <p>Note: Note that count is a byte count, this facilitates comparing word aligned byte arrays when using word aligned packed strings.</p> <p>Source can be a string constant e.g. <code>myptr := mem_Copy("TEST STRING", ptr2, 12);</code></p>	
Example	<code>myptr := mem_Copy(ptr1, ptr2, 100);</code>	

2.19.9. mem_Compare(ptr1, ptr2, count)

Syntax	<code>mem_Compare(ptr1, ptr2, count);</code>	
Arguments	ptr1, ptr2, count	
	ptr1	Specifies the 1st memory block.
	ptr2	Specifies the 2nd memory block.
	count	Specifies the number of bytes to compare.
Returns	Value	
	Value	Returns 0 if we have a match, -1 if ptr1 < ptr2, and +1 if ptr2 > ptr1. (The comparison is done alphabetically)
Description	Compare two blocks of memory ptr1 and ptr2 .	
Example	<code>test := mem_Compare(this_block, that_block, 100);</code>	

2.19.10. mem_ArrayOp1(memarray, count, op, value)

Syntax	<code>mem_ArrayOp1(memarray, count, op, value);</code>																																																										
Arguments	<code>memarray, count, op, value</code>																																																										
	memarray	Pointer to the array to be operated on																																																									
	count	Size of the array																																																									
	op	One of the constants defining the operation to be performed (see below)																																																									
	value	Value that may be required by the selected operation																																																									
Returns	None																																																										
Description	<p>This function (and the similar mem_ArrayOp2 function) can be used to perform highly optimised operation against an array of data. Mem_ArrayOp1 is for Single Arrays.</p> <p>Single Word Array Operations:</p> <table> <tr><td>OP1_NOP</td><td>0</td><td>// no operation</td></tr> <tr><td>OP1_SET</td><td>1</td><td>// "set" the entire array with "value"</td></tr> <tr><td>OP1_AND</td><td>2</td><td>// "and" the entire array with "value"</td></tr> <tr><td>OP1_IOR</td><td>3</td><td>// "inclusive or" the entire array with "value"</td></tr> <tr><td>OP1_XOR</td><td>4</td><td>// "exclusive or" the entire array with "value"</td></tr> <tr><td>OP1_ADD</td><td>5</td><td>// signed add each element of entire array with "value"</td></tr> <tr><td>OP1_SUB</td><td>6</td><td>// signed subtract "value" from each element of entire array.</td></tr> <tr><td>OP1_MUL</td><td>7</td><td>// signed multiply each element of entire array by "value"</td></tr> <tr><td>OP1_DIV</td><td>8</td><td>// signed divide each element of entire array by "value"</td></tr> <tr><td>OP1_REV</td><td>9</td><td>// reverse the elements of an array (value is ignored)</td></tr> <tr><td>OP1_SHL</td><td>10</td><td>// shift an array left by "value" positions</td></tr> <tr><td>OP1_SHR</td><td>11</td><td>// shift an array right by "value" positions</td></tr> <tr><td>OP1_ROL</td><td>12</td><td>// rotate an array left by "value" positions</td></tr> <tr><td>OP1_ROR</td><td>13</td><td>// rotate an array right by "value" positions</td></tr> </table> <p>Graphics only Operations:</p> <table> <tr><td>OP1_GRAY</td><td>14</td><td>// convert an array of RGB565 elements to grayscale, "value" is ignored</td></tr> <tr><td>OP1_WHITEN</td><td>15</td><td>// saturate an array of RGB565 elements to white, "value" determines saturation</td></tr> <tr><td>OP1_BLACKEN</td><td>16</td><td>// saturate an array of RGB565 elements to black, "value" determines saturation</td></tr> <tr><td>OP1_LIGHTEN</td><td>17</td><td>// increase luminance of an array of RGB565 elements, "value" determines saturation</td></tr> <tr><td>OP1_DARKEN</td><td>18</td><td>// decrease luminance of an array of RGB565 elements, "value" determines saturation</td></tr> </table>		OP1_NOP	0	// no operation	OP1_SET	1	// "set" the entire array with "value"	OP1_AND	2	// "and" the entire array with "value"	OP1_IOR	3	// "inclusive or" the entire array with "value"	OP1_XOR	4	// "exclusive or" the entire array with "value"	OP1_ADD	5	// signed add each element of entire array with "value"	OP1_SUB	6	// signed subtract "value" from each element of entire array.	OP1_MUL	7	// signed multiply each element of entire array by "value"	OP1_DIV	8	// signed divide each element of entire array by "value"	OP1_REV	9	// reverse the elements of an array (value is ignored)	OP1_SHL	10	// shift an array left by "value" positions	OP1_SHR	11	// shift an array right by "value" positions	OP1_ROL	12	// rotate an array left by "value" positions	OP1_ROR	13	// rotate an array right by "value" positions	OP1_GRAY	14	// convert an array of RGB565 elements to grayscale, "value" is ignored	OP1_WHITEN	15	// saturate an array of RGB565 elements to white, "value" determines saturation	OP1_BLACKEN	16	// saturate an array of RGB565 elements to black, "value" determines saturation	OP1_LIGHTEN	17	// increase luminance of an array of RGB565 elements, "value" determines saturation	OP1_DARKEN	18	// decrease luminance of an array of RGB565 elements, "value" determines saturation
OP1_NOP	0	// no operation																																																									
OP1_SET	1	// "set" the entire array with "value"																																																									
OP1_AND	2	// "and" the entire array with "value"																																																									
OP1_IOR	3	// "inclusive or" the entire array with "value"																																																									
OP1_XOR	4	// "exclusive or" the entire array with "value"																																																									
OP1_ADD	5	// signed add each element of entire array with "value"																																																									
OP1_SUB	6	// signed subtract "value" from each element of entire array.																																																									
OP1_MUL	7	// signed multiply each element of entire array by "value"																																																									
OP1_DIV	8	// signed divide each element of entire array by "value"																																																									
OP1_REV	9	// reverse the elements of an array (value is ignored)																																																									
OP1_SHL	10	// shift an array left by "value" positions																																																									
OP1_SHR	11	// shift an array right by "value" positions																																																									
OP1_ROL	12	// rotate an array left by "value" positions																																																									
OP1_ROR	13	// rotate an array right by "value" positions																																																									
OP1_GRAY	14	// convert an array of RGB565 elements to grayscale, "value" is ignored																																																									
OP1_WHITEN	15	// saturate an array of RGB565 elements to white, "value" determines saturation																																																									
OP1_BLACKEN	16	// saturate an array of RGB565 elements to black, "value" determines saturation																																																									
OP1_LIGHTEN	17	// increase luminance of an array of RGB565 elements, "value" determines saturation																																																									
OP1_DARKEN	18	// decrease luminance of an array of RGB565 elements, "value" determines saturation																																																									
Example	<pre>var a1[20] ; func dumpA1d(var cnt) var i ;</pre>																																																										

```
    for (i := 0; i < cnt; i++)
        print([DEC5ZB] a1[i], " ") ;
    next
    print("\n") ;
endfunc

func main()
    var i, j, res[2], v1[2], v2[2] ;

    a1[0] := 100; a1[1] := 1000 ; a1[2] := 10000 ; a1[3] := 40000 ;
    dumpA1d(4) ;
    print("ADD ") ;
    mem_ArrayOp1(a1, 4, OP1_ADD, 10) ;
    dumpA1d(4) ;
    a1[0] := 100; a1[1] := 1000 ; a1[2] := 10000 ; a1[3] := 40000 ;
    print("SUB ") ;
    mem_ArrayOp1(a1, 4, OP1_SUB , 10) ;
    dumpA1d(4) ;
    a1[0] := 100; a1[1] := 1000 ; a1[2] := 10000 ; a1[3] := 40000 ;
    print("MUL ") ;
    mem_ArrayOp1(a1, 4, OP1_MUL, 10) ;
    dumpA1d(4) ;
    a1[0] := 100; a1[1] := 1000 ; a1[2] := 10000 ; a1[3] := 40000 ;
    print("DIV ") ;
    mem_ArrayOp1(a1, 4, OP1_DIV, 10) ;
    dumpA1d(4) ;
    a1[0] := 100; a1[1] := 1000 ; a1[2] := 10000 ; a1[3] := 40000 ;
    print("REV ") ;
    mem_ArrayOp1(a1, 4, OP1_REV, 10) ;
    dumpA1d(4) ;

    repeat
        forever
    endfunc
```

2.19.11. mem_ArrayOp2(memarray1, memarray2, count, op, value)

Syntax	mem_ArrayOp1(memarray1, memarray2, count, op, value);	
Arguments	memarray1, memarray2, count, op, value	
	memarray1	Pointer to the 1 st array to be operated on
	memarray2	Pointer to the 2 nd array to be operated on
	count	Size of the array
	op	One of the constants defining the operation to be performed (see below)
	value	Value that may be required by the selected operation
Returns	None	
Description	<p>This function (and the similar mem_ArrayOp2 function) can be used to perform highly optimised operation against an array of data. Mem_ArrayOp2 is for Dual Arrays.</p> <p>Boolean and Maths Opeations:</p> <pre>OP2_AND 1 // "and" arrays, result to array1 (value is ignored) OP2_IOR 2 // "inclusive or" arrays, result to array1 (value is ignored) OP2_XOR 3 // "exclusive or" arrays, result to array1 (value is ignored) OP2_ADD 4 // "add" arrays, result to array1, array1 + (array2+value) OP2_SUB 5 // "subtract" array2 from array1, result to array1, array1-(array2+value) OP2_MUL 6 // "multiply" arrays, result to array1 (value is ignored) OP2_DIV 7 // "divide array1 by array2" , result to array1 (value is ignored) OP2_COPY 8 // "copy" array2 to array1 (value is ignored)</pre> <p>Graphics only Operations:</p> <pre>OP2_BLEND 9 // blend arrays, blend percentage determined by "value", result to "array1"</pre>	
Example		

2.20. General Purpose Functions

Summary of Functions in this section:

- pause(time)
- lookup8 (**key**, byteConstList)
- lookup16 (**key**, wordConstList)

2.20.1. pause(time)

Syntax	pause(time);	
Arguments	time	
	time	A value specifying the delay time in milliseconds.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Stop execution of the user program for a predetermined amount of time.	
Example	<pre> if (status) // if fire button pressed pause(30) // slow down the loop else ... </pre>	

2.20.2. lookup8(key, byteConstList)

Syntax	<code>lookup8(key, byteConstList);</code>	
Arguments	key, byteConstList	
	key	A byte value to search for in a fixed list of constants. The key argument can be a variable, array element, expression or constant
	byteConstList	A comma separated list of constants and strings to be matched against key . Note: the string of constants may be freely formed, see example.
Returns	result	
	result	See description.
Description	<p>Search a list of 8 bit constant values for a match with a search value key. If found, the index of the matching constant is returned in result, else result is set to zero. Thus, if the value is found first in the list, result is set to one. If second in the list, result is set to two etc. If not found, result is returned with zero.</p> <p>Note: The list of constants cannot be re-directed. The <code>lookup8(...)</code> functions offer a versatile way for returning an index for a given value. This can be very useful for data entry filtering and parameter input checking and where ever you need to check the validity of certain inputs. The entire search list field can be replaced with a single name if you use the \$ operator in constant, eg:</p> <pre>#constant HEXVALUES \$"0123456789ABCDEF"</pre>	
Example	<pre>func main() var key, r; key := 'a'; r := lookup8(key, 0x4D, "abcd", 2, 'Z', 5); print("\nSearch value 'a' \nfound as index ", r) key := 5; r := lookup8(key, 0x4D, "abcd", 2, 'Z', 5); print("\nSearch value 5 \nfound at index ", r) putstr("\nScanning..\n"); key := -12000; // we will count from -12000 to +12000, only // the hex ascii values will give a match value while(key <= 12000) r := lookup8(key, "0123456789ABCDEF"); // hex lookup if(r) print([HEX1] r-1); // only print if we got a match in // the table key++; wend repeat forever endfunc</pre>	

2.20.3. lookup16(key, wordConstList)

Syntax	<code>lookup16(key, wordConstList);</code>	
Arguments	<code>key, wordConstList</code>	
	key	A word value to search for in a fixed list of constants. The key argument can be a variable, array element, expression or constant
	wordConstList	A comma separated list of constants to be matched against key .
Returns	result	
	result	See description.
Description	<p>Search a list of 16 bit constant values for a match with a search value key. If found, the index of the matching constant is returned in result, else result is set to zero. Thus, if the value is found first in the list, result is set to one. If second in the list, result is set to two etc. If not found, result is returned with zero.</p> <p>Note: The <code>lookup16(...)</code> functions offer a versatile way for returning an index for a given value. This is very useful for parameter input checking and where ever you need to check the validity of certain values. The entire search list field can be replaced with a single name by using the \$ operator in constant, eg:</p> <pre>#constant LEGALVALS \$5,10,20,50,100,200,500,1000,2000,5000,10000</pre>	
Example	<pre>func main() var key, r; key := 5000; r := lookup16(key, 5,10,20,50,100,200,500,1000,2000,5000,10000); //r := lookup16(key, LEGALVALS); if(r) print("\nSearch value 5000 \nfound at index ", r); else putstr("\nValue not found"); endif print("\nOk"); // all done repeat forever endfunc</pre>	

2.21. Floating point Functions

Summary of Functions in this section:

- `flt_ADD(&result, &floatA, &floatB)`
- `flt_SUB(&result, &floatA, &floatB)`
- `flt_MUL(&result, &floatA, &floatB)`
- `flt_DIV(&result, &floatA, &floatB)`
- `flt_POW(&result, &floatA, &floatB)`
- `flt_ABS(&result, &floatval)`
- `flt_CEIL(&result, &floatval)`
- `flt_FLOOR(&result, &floatval)`
- `flt_SIN(&result, &floatval)`
- `flt_COS(&result, &floatval)`
- `flt_TAN(&result, &floatval)`
- `flt_ASIN(&result, &floatval)`
- `flt_ACOS(&result, &floatval)`
- `flt_ATN(&result, &floatval)`
- `flt_EXP(&result, &floatval)`
- `flt_LOG(&result, &floatval)`
- `flt_SQR(&result, &floatval)`
- `flt_LT(&floatA, &floatB)`
- `flt_EQ(&floatA, &floatB)`
- `flt_NE(&floatA, &floatB)`
- `flt_GT(&floatA, &floatB)`
- `flt_GE(&floatA, &floatB)`
- `flt_LE(&floatA, &floatB)`
- `flt_SGN(&floatval)`
- `flt_FTOI(&floatval)`
- `flt_ITOF(&fresult, &var16)`
- `flt_UITOF(&fresult, &uvar16)`
- `flt_LTOF(&fresult, &var32)`
- `flt_ULTOF(&fresult, &uvar32)`
- `flt_VAL(&float1, mystring)`
- `flt_PRINT(&fvalue, formatstring)`
- `flt_PRINTxy(x, y, &fvalue, formatstring)`

2.21.1. **flt_ADD(&result, &floatA, &floatB)**

Syntax	flt_ADD(&result, &floatA, &floatB)	
Arguments	&result, &floatA, &floatB	
	&result	Points to float result register.
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling flt_Val for a string argument.	
	Note: A float variable is a 2 word array, eg var myfloat[2]	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Performs floating point addition (A+B) and returns the value in the result register.	
Example		

2.21.2. `flt_SUB(&result, &floatA, &floatB)`

Syntax	<code>flt_SUB(&result, &floatA, &floatB)</code>	
Arguments	<code>&result, &floatA, &floatB</code>	
	&result	Points to float result register.
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	<code>pointer</code>	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Performs floating point Subtraction (A-B) and returns the value in the result register.	
Example		

2.21.3. `flt_MUL(&result, &floatA, &floatB)`

Syntax	<code>flt_MUL(&result, &floatA, &floatB)</code>	
Arguments	<code>&result, &floatA, &floatB</code>	
	&result	Points to float result register.
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	<code>pointer</code>	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Performs floating point Multiplication ($A * B$) and returns the value in the result register.	
Example		

2.21.4. `flt_DIV(&result, &floatA, &floatB)`

Syntax	<code>flt_DIV(&result, &floatA, &floatB)</code>	
Arguments	<code>&result, &floatA, &floatB</code>	
	&result	Points to float result register.
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	<code>pointer</code>	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Performs floating point Division (A/B) and returns the value in the result register.	
Example		

2.21.5. **flt_POW(&result, &floatA, &floatB)**

Syntax	flt_POW(&result, &floatA, &floatB)	
Arguments	&result, &floatA, &floatB	
	&result	Points to float result register.
	&floatA	Points to the float value to raise.
	&floatB	Points to the float value for power.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling flt_Val for a string argument.	
	Note: A float variable is a 2 word array, eg var myfloat[2]	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Raises A to power B (A^B) and returns the result value in the result register.	
Example		

2.21.6. `flt_ABS(&result, &floatval)`

Syntax	<code>flt_ABS(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value to get the Absolute of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates absolute value of the floating point value.	
Example		

2.21.7. `flt_CEIL(&result, &floatval)`

Syntax	<code>flt_CEIL(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value to integerize up.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Rounds value up to the integer value. Removes fractional part, rounding up correctly.	
Example		

2.21.8. flt_FLOOR(&result, &floatval)

Syntax	flt_FLOOR(&result, &floatval)	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value to integerize down.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling flt_Val for a string argument.	
	Note: A float variable is a 2 word array, eg var myfloat[2]	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Rounds value down to the integer value. Removes fractional part, rounding down correctly.	
Example		

2.21.9. `flt_SIN(&result, &floatval)`

Syntax	<code>flt_SIN(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	&floatval	Points to the float value angle (in radians) to get the SINE of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the SINE of float value in radians and returns the value in the result register.	
Example		

2.21.10. `flt_COS(&result, &floatval)`

Syntax	<code>flt_COS(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value angle (in radians) to get the COSINE of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the COSINE of float value in radians and returns the value in the result register.	
Example		

2.21.11. flt_TAN(&result, &floatval)

Syntax	flt_TAN(&result, &floatval)	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value angle (in radians) to get the TANGENT of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling flt_Val for a string argument.	
	Note: A float variable is a 2 word array, eg var myfloat[2]	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the TANGENT of float value in radians and returns the value in the result register.	
Example		

2.21.12. `flt_ASIN(&result, &floatval)`

Syntax	<code>flt_ASIN(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register. Result is in radians.
	&floatval	Points to the float value to get the ARCSINE of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the ARCSINE of float value and returns the angle in radians in the result register.	
Example		

2.21.13. `flt_ACOS(&result, &floatval)`

Syntax	<code>flt_ACOS(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register. Result is in radians.
	&floatval	Points to the float value to get the ARCCOS of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the ARCCOS of float value and returns the angle in radians in the result register.	
Example		

2.21.14. `flt_ATAN(&result, &floatval)`

Syntax	<code>flt_ATAN(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register. Result is in radians.
	&floatval	Points to the float value to get the ARCTAN of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the ARCTAN of float value and returns the angle in radians in the result register.	
Example		

2.21.15. `flt_EXP(&result, &floatval)`

Syntax	<code>flt_EXP(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value to get the Exponent of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the Exponent of float value and returns the value in the result register.	
Example		

2.21.16. `flt_LOG(&result, &floatval)`

Syntax	<code>flt_LOG(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value to get the natural Log of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the Natural Log of float value and returns the value in the result register.	
Example		

2.21.17. `flt_SQR(&result, &floatval)`

Syntax	<code>flt_SQR(&result, &floatval)</code>	
Arguments	&result, &floatval	
	&result	Points to float result register.
	& floatval	Points to the float value to get the square root of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns a pointer to the float result register or zero if error occurs. Carry and overflow are not affected.
Description	Calculates the square root of float value and returns the value in the result register.	
Example		

2.21.18. `flt_LT(&floatA, &floatB)`

Syntax	<code>flt_LT(&floatA, &floatB)</code>	
Arguments	& floatA, &floatB	
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	True if A < B, false otherwise
Description	Compare A to B and returns true if A < B	
Example		

2.21.19. `flt_EQ(&floatA, &floatB)`

Syntax	<code>flt_EQ(&floatA, &floatB)</code>	
Arguments	& floatA, &floatB	
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	True if <code>A == B</code> , false otherwise
Description	Compare A to B and returns true if equal.	
Example		

2.21.20. `flt_NE(&floatA, &floatB)`

Syntax	<code>flt_NE(&floatA, &floatB)</code>	
Arguments	& floatA, &floatB	
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	True if A != B, false otherwise
Description	Compare A to B and returns true if A != B	
Example		

2.21.21. `flt_GT(&floatA, &floatB)`

Syntax	<code>flt_GT(&floatA, &floatB)</code>	
Arguments	& floatA, &floatB	
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	True if A > B, false otherwise
Description	Compare A to B and returns true if A > B	
Example		

2.21.22. `flt_GE(&floatA, &floatB)`

Syntax	<code>flt_GE(&floatA, &floatB)</code>	
Arguments	& floatA, &floatB	
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	True if $A \geq B$, false otherwise
Description	Compare A to B and returns true if $A \geq B$.	
Example		

2.21.23. `flt_LE(&floatA, &floatB)`

Syntax	<code>flt_LE(&floatA, &floatB)</code>	
Arguments	& floatA, &floatB	
	&floatA	Points to the float value A.
	&floatB	Points to the float value B.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	True if $A \leq B$, false otherwise
Description	Compare A to B and returns true if $A \leq B$	
Example		

2.21.24. `flt_SGN(&floatval)`

Syntax	<code>flt_SGN(&floatval)</code>	
Arguments	& floatval	
	&floatval	Points to the float value to examine the sign of.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	value	
	value	Returns 16bit integer -1 if float sign is negative, or zero if positive
Description	Examines sign of the float value and returns 0 if sign is positive or value equals zero. Returns 16bit integer -1 if float sign is negative	
Example		

2.21.25. flt_FTOI(&floatval)

Syntax	flt_FTOI(&floatval)	
Arguments	& floatval	
	&floatval	Points to the float value to be converted to integer.
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling flt_Val for a string argument.	
	Note: A float variable is a 2 word array, eg var myfloat[2]	
Returns	value	
	value	The integer value of the float
Description	Converts a floating point number to a 16bit integer. The floating point number is rounded up or down accordingly.	
Example		

2.21.26. flt_ITOF(&fresult, var16)

Syntax	flt_ITOF(&fresult, var16)	
Arguments	&fresult, var16	
	&fresult	Points to float result variable.
	var16	a 16bit signed integer variable
	Note: A float variable is a 2 word array, eg var fresult[2]	
Returns	pointer	
	pointer	Returns the pointer to the float result, normally ignored
Description	Converts a 16bit signed integer value to a signed floating point number.	
Example		

2.21.27. flt_UITOF(&fresult, uvar16)

Syntax	flt_UITOF(&fresult, uvar16)	
Arguments	&fresult, uvar16	
	&fresult	Points to float result variable.
	uvar16	A 16bit unsigned integer variable
	Note: A float variable is a 2 word array, eg var fresult[2]	
Returns	pointer	
	pointer	Returns the pointer to the float result.
Description	Converts a 16bit unsigned integer value to a positive floating point number.	
Example		

2.21.28. flt_LTOF(&fresult, var32)

Syntax	flt_LTOF(&fresult, var32)	
Arguments	&fresult, var32	
	&fresult	Points to float result variable.
	var32	A 32bit (long) signed variable.
	Note: A float variable is a 2 word array, eg var fresult[2]	
Returns	pointer	
	pointer	Returns the pointer to the float result.
Description	Converts a 32bit signed integer value to a signed floating point number.	
Example		

2.21.29. flt_ULTOF(&fresult, uvar32)

Syntax	flt_ULTOF(&fresult, uvar32)	
Arguments	&fresult, uvar32	
	&fresult	Points to float result variable.
	uvar32	A 32bit (long) unsigned variable.
	Note: A float variable is a 2 word array, eg var fresult[2]	
Returns	pointer	
	pointer	Returns the pointer to the float result.
Description	Converts a 32bit unsigned integer value to a positive floating point number.	
Example		

2.21.30. `flt_VAL(&fresult, numstring)`

Syntax	<code>flt_VAL(&fresult, numstring)</code>	
Arguments	&fresult, numstring	
	& fresult	Points to float result register.
	numstring	A string constant or string variable that holds valid floating point number. The string argument can be a string constant, a pointer to a string variable, or a pointer to a data statement. The string may be a float, or a hex or binary integer value (no decimal point allowed). For hex or binary, the number is preceded with 0x or 0b
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	pointer	
	pointer	Returns the pointer to the float result.
Description	Converts the number string to a valid float value. Carry and overflow are not affected.	
Example		

2.21.31. **flt_PRINT (&fvalue, formatstring)**

Syntax	flt_PRINT(&fvalue, formatstring)	
Arguments	&fvalue, formatstring	
	&fvalue	Points to float result variable.
	formatstring	zero, null string, of valid format string
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling <code>flt_Val</code> for a string argument.	
	Note: A float variable is a 2 word array, eg <code>var myfloat[2]</code>	
Returns	status	
	status	Returns '0' if successfull.
Description	Prints a floating point value in a set string format. The string argument can be a string constant, a pointer to a string variable, or a pointer to a data statement. If it is zero or an empty string, the number is automatically formatted for the best presentation. The format string is similar to the C language, but only a single '%' may be used to print a single variable.	
Example		

2.21.32. flt_PRINTxy (x, y, &fvalue, formatstring)

Syntax	flt_PRINTxy(x, y, &fvalue, formatstring)	
Arguments	x, y, &fvalue, formatstring	
	x	The x position to start printing the number in.
	y	The y position to start printing the number in.
	&fvalue	Points to float result variable.
	formatstring	zero, null string, of valid format string
	Arguments may be a pointer to a float variable or a numeric text string. A string argument is converted at run-time by calling flt_Val for a string argument.	
	Note: A float variable is a 2 word array, eg var myfloat[2]	
Returns	Status	
	status	Returns '0' if successfull.
Description	Prints a floating point value in a set string format at the specified position. The string argument can be a string constant, a pointer to a string variable, or a pointer to a data statement. If it is zero or an empty string, the number is automatically formatted for the best presentation. The format string is similar to the C language, but only a single '%' may be used to print a single variable.	
Example		

2.22. Misc System Functions

Summary of Functions in this section:

- `sys_PmmC()`
- `sys_Driver()`

2.22.1. sys_Pmmc()

Syntax	<code>sys_Pmmc();</code>
Arguments	None
Returns	None
Description	Prints the system Pmmc name and revision eg "Diablo16\n1.0" Can be captured to a buffer using the to() function
Example	<pre>to(myString); sys_Pmmc();// save Pmmc name and revision to buffer</pre>

2.22.2. sys_Driver()

Syntax	<code>sys_Driver();</code>
Arguments	None
Returns	None
Description	Prints the system driver name and date string eg "uLCD-32WDTU-A\n130411" Can be captured to a buffer using the to() function
Example	<pre>to(mystring); sys_Driver(); // save Driver name and date to buffer</pre>

2.23. SPI FLASH Functions

Summary of Functions in this section:

- spiflash_BlockErase(spi#, Enablepin, block)
- spiflash_BulkErase(spi#, Enablepin)
- spiflash_Exec(spi#, Enablepin, arglistptr)
- spiflash_GetC(spi#, Enablepin)
- spiflash_GetS(*String, size, spi#, Enablepin)
- spiflash_GetW(spi#, Enablepin)
- spiflash_ID(spi#, Enablepin)
- spiflash_Image(x, y, spi#, Enablepin)
- spiflash_LoadFunction(spi#, Enablepin)
- spiflash_LoadImageControl(spi#, Enablepin)
- spiflash_PlayWAV(spi#, Enablepin)
- spiflash_PutC(char, spi#, Enablepin)
- spiflash_PutS(source, spi#, Enablepin)
- spiflash_PutW(word, spi#, Enablepin)
- spiflash_Read(destination, size, spi#, Enablepin)
- spiflash_Run(spi#, Enablepin, arglistptr)
- spiflash_SetAdd(spi#, HiWord, LoWord)
- spiflash_SIG(spi#, Enablepin)
- spiflash_Write(Source, size, spi#, Enablepin)

These functions can be used to access an SPI FLASH storage device connected to the selected SPI port, and correctly initialised with the spi_Init(...) function, each FLASH device also needs a dedicated enable pin pulled high and set as output from within the driving program. The only devices supported so far are the M25Pxx and A25Lxx range of devices which are 512Kbit to 128Mbit Serial Flash Memory. Other similar devices should also work, provided they do not exceed 128Mbit.

Note that when accessing certain file types via spiflash it may be necessary to append an identifiable EOF character (eg ^Z) to enable your program to properly detect EOF.

Sample initialization code:-

```
#CONST
  EnablePin PA0
  ClockPin PA6
  SDIPin PA2
  SDOPin PA5
#END

pin_HI(EnablePin) ;
pin_Set(PIN_OUT,EnablePin) ;
SPI1_SDI_pin(SDIPin) ;
SPI1_SCK_pin(ClockPin) ;
SPI1_SDO_pin(SDOPin) ;
SPI1_Init(SPI_SPEED15, SPI8_MODE_5) ;
```

Note that the Init must be done in 8 bit mode, but the internal functions will automatically flip between 8 and 16 bit mode to gain optimal performance.

2.23.1. spiflash_BlockErase(spi#, Enablepin, block)

Syntax	<code>spiflash_BlockErase(spi#, Enablepin, block) ;</code>	
Arguments	<code>spi#, Enablepin, block</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
	Block	The block to be erased
Returns	Nothing	
Description	Erases the required block in a FLASH media device. The function returns no value, and the operation can take up to 3 milliseconds.	
Example	<code>spiflash_BlockErase(SPI1, PA0, 3) ;</code>	

2.23.2. spiflash_BulkErase(spi#, Enablepin)

Syntax	<code>spiflash_BulkErase(spi#, Enablepin) ;</code>	
Arguments	<code>spi#, Enablepin</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Nothing	
Description	Erases the entire flash media device. The function returns no value, and the operation can take up to 80 seconds depending on the size of the flash device. Note that not all devices support this command.	
Example	<code>spiflash_BulkErase(SPI1, PA0) ;</code>	

2.23.3. spiflash_Exec(spi#, Enablepin, arglistptr)

Syntax	<code>spiflash_Exec(spi#, Enablepin, arglistptr);</code>	
Arguments	<code>spi#, Enablepin, arglistptr</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
	arglistptr	pointer to the list of arguments to pass to the new program or 0 if no arguments.
Returns	Value	
	Value	Returns the value from main in the called program.
Description	<p>This function is similar to spiflash_Run, however, the main program in FLASH retains all memory allocations (eg file buffers, memory allocated with mem_Alloc etc)</p> <p>Returns like a function, current program calling program is kept active and control returns to it.</p> <p>If arglistptr is 0, no arguments are passed, else arglist points to an array, the first element being the number of elements in the array.</p> <p>func 'main' in the called program accepts the arguments.</p> <p>This function is similar to spiflash_LoadFunction(...), however, the function argument list is passed by pointer, and the memory consumed by the function is released as soon as the function completes.</p> <p>spiflash_SetAdd should have previously been called to identify the address of the program to be called.</p>	
Example	<code>spiflash_Exec(SPI1, PA0, 0) ;</code>	

2.23.4. spiflash_GetC(spi#, Enablepin)

Syntax	1.1.1. spiflash_GetC(spi#, Enablepin);	
Arguments	spi#, Enablepin	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	byte	
	byte	Returns the next char from the file
Description	Reads a character (or byte) from the SPI FLASH memory device on the specified SPI port and enable pin. The source is the address set by spiflash_SetAdd() , or incremented by subsequent reads or writes.	
Example	<code>mychar := spiflash_GetC(SPI1, PA0) ;</code>	

2.23.5. spiflash_GetS(*String, size, spi#, Enablepin)

Syntax	<code>spiflash_GetS(*String, size, spi#, Enablepin) ;</code>	
Arguments	<code>string, size, spi#, Enablepin</code>	
	string	Destination buffer
	size	The maximum number of bytes to be read from the file. (Up to max of 80)
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Count	
	Count	Returns the number of characters read from file (excluding the null terminator)
Description	<p>This function Reads a line of text to a buffer (specified by "*string") from the FLASH memory device on the specified SPI port and enable pin into the specified destination. The source is the address set by <code>spiflash_SetAdd()</code>, or incremented by subsequent reads or writes.</p> <p>Note: only reads up to "size-1" characters into "string"</p> <p><code>file_GetS(...)</code> will stop reading when any of the following conditions are true:</p> <ul style="list-style-type: none"> A) It has read n-1 bytes (one character is reserved for the null-terminator) B) It encounters a newline character (a line-feed in the compilers tested here) C) It reaches the end of file D) A read error occurs. <p>The file must be previously opened with 'r' (read) mode.</p>	
Example	<code>res := spiflash_GetS(mystring, 80, SPI1, PA0);</code>	

2.23.6. spiflash_GetW(spi#, Enablepin)

Syntax	<code>spiflash_GetW(spi#, Enablepin);</code>	
Arguments	<code>spi#, Enablepin</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Word	
	Word	Returns the next word in the file
Description	This function reads a word (2 bytes) from the FLASH memory device on the specified SPI port and enable pin, at the <code>spiflash_SetAdd()</code> , or incremented by subsequent reads or writes and advances the pointer appropriately (incremented by 2).	
Example	<code>myword := spiflash_GetW(hndl);</code>	

2.23.7. spiflash_ID(spi#, Enablepin)

Syntax	<code>spiflash_ID(spi#, Enablepin) ;</code>	
Arguments	<code>spi#, Enablepin</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Nothing	
Description	Reads the memory type and capacity from the serial FLASH device. Hi byte contains type, and low byte contains capacity. Refer to the device data sheet for further information.	
Example	<code>Id := spiflash_ID(SPI1, PA0) ;</code>	

2.23.8. spiflash_Image(x, y, spi#, Enablepin)

Syntax	<code>spiflash_Image(x, y, spi#, Enablepin) ;</code>	
Arguments	<code>x, y, spi#, Enablepin</code>	
	x	X-position of the image to be displayed
	Y	Y-position of the image to be displayed
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Returns a copy of the <code>file_Error()</code> error code	
Description	Display an image from the SPI FLASH at screen location specified by x, y(top left corner). The image is displayed from a file at the current FLASH position set by <code>spiflash_SetAdd()</code> .	
Example	<code>spiflash_Image(x, y, SPI1, PA0) ;</code>	

2.23.9. spiflash_LoadFunction(spi#, Enablepin)

Syntax	spiflash_LoadFunction(spi#, Enablepin)	
Arguments	spi#, Enablepin	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Pointer	
	Pointer	Returns a pointer to the memory allocation where the function has been loaded from file which can be then used as a function call.
Description	<p>Load a function or program from the FLASH memory device on the specified SPI port and enable pin at the address set by spiflash_SetAdd(), or incremented by subsequent reads or writes and return a function pointer to the allocation.</p> <p>The function can then be invoked just like any other function would be called via a function pointer. Parameters may be passed to it in a conventional way. The function may be discarded at any time when no longer required, thus freeing its memory resources.</p> <p>The loaded function can be discarded with mem_Free(..) Note that any pointer references passed to the child function may not include references to the parents DATA statements or any static string references. Any string or array information must be in the parents global or local memory space. The reason for this is that DATA statements and static strings are contained in the parents CODE segment, and cannot be accessed by the child process.</p> <p>The callers stack is shared by the loaded function, however any global variables in the loaded function are private to that function.</p>	
Example1	<pre>var titlestring[20]; var textstring[20]; to(titlestring); putstr("My Window Title"); to(textstring); putstr("My Special Message"); popupWindow := spiflash_LoadFunction(SPI1, PA0); if(!popupWindow)goto LoadFunctionFailed; //could not load the function //then elsewhere in your program res := popupWindow(MYMODE,titlestring,textstring); if(res == QUIT APPLICATION) goto exitApp; //Later in your program, when popupWindow is no longer required //for the application res := mem_Free(popupWindow); if(!res) goto FreeFunctionFailed; //should never happen if memory not //corrupted</pre>	
Example2	<pre>var fncHandle; //a var for a handle to sliders2.4dg var slidervals; //reference var to access global vars in sliders.4dg fncHandle := spiflash_LoadFunction(SPI1, PA0); // load the function slidervals := fncHandle&0x7FFF; // note that memory allocations for transient programs are biased with 8000h which must be removed. slidervals++; // note that all globals start at '1'</pre>	

```
slidervals[0] := 25; // set sliders to initial positions
slidervals[1] := 20;
slidervals[2] := 30;
slidervals[3] := 15;
slidervals[4] := 35;
slidervals[5] := 20;
slidervals[6] := 40;
slidervals[7] := 25;
slidervals[8] := 45;
slidervals[9] := 5;

r := fncHandle(); // activate the function

print("Return value = 0x", [HEX] r, "\n");

// print the values, they may have changed
print("Slider 1 ", slidervals[0], " Slider 2 ", slidervals[1], "\n");
print("Slider 3 ", slidervals[2], " Slider 4 ", slidervals[3], "\n");
print("Slider 5 ", slidervals[4], " Slider 6 ", slidervals[5], "\n");
print("Slider 7 ", slidervals[6], " Slider 8 ", slidervals[7], "\n");
print("Slider 9 ", slidervals[8], " Slider 10 ", slidervals[9], "\n");

mem_Free(fncHandle); // done with sliders, release its memory
```

2.23.10. spiflash_LoadImageControl(spi#, Enablepin)

Syntax	<code>spiflash_LoadImageControl(spi#, Enablepin) ;</code>	
Arguments	spi#, Enablepin	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Status	
	Status	Returns a handle (pointer to the memory allocation) to the image control list that has been created. Returns NULL if function fails.
Description	<p>Reads a control file to create an image list from the FLASH memory device on the specified SPI port and enable pin. The source is the address set by <code>spiflash_SetAdd()</code>, or incremented by subsequent reads or writes. The ".dat" file is first and is immediately followed by a ^Z and then the ".gci" file.</p> <p>When an image control is loaded, an array is built in ram. It consists of a 6 word header with the following entries as defined by the constants:</p> <pre> IMG_COUNT 0 IMG_ENTRYLEN 1 IMG_MODE 2 IMG_GCI_FILENAME 3 IMG_DAT_FILENAME 4 IMG_GCIFILE_HANDLE 5 </pre> <p>No images are stored in FLASH or RAM, the image control holds the index values for the absolute storage positions on the uSD card for RAW mode, or the cluster/sector position for formatted FAT16 mode.</p> <p>When an image control is no longer required, the memory can be released with:</p> <pre>mem_Free(MyImageControlHandle);</pre>	
Example	<pre> #inherit "4DGL_16bitColours.fnc" #constant OK 1 #constant FAIL 0 var p; // buffer pointer var img; // handle for the image list var n, exit, r; //----- // return true if screen touched, also sets ok flag func CheckTouchExit() return (exit := (touch_Get(TOUCH_STATUS) == TOUCH_PRESSED)); // if there's a press, exit endfunc //----- </pre>	

```

func main()

    gfx_Cls();
    txt_Set(FONT_ID, FONT_2);
    txt_Set(TEXT_OPACITY, OPAQUE);

    touch_Set(TOUCH_ENABLE);           // enable the touch screen

    print("heap=", mem_Heap(), " bytes\n"); // show the heap size

    r := OK; // return value
    exit := 0;

    if (!file_Mount())
        print("File error ", file_Error());
        while(!CheckTouchExit());
// just hang if we didnt get the image list
        r := FAIL;
        goto quit;
    endif

    print ("WAIT...building image list\n");

    // slow build, fast execution, higher memory requirement
    img := spiflash_LoadImageControl(SPI1, PA0);
    // build image control, returning a pointer to structure allocation

    if (img)
        print("image control=", [HEX] img, "\n");
// show the address of the image control allocation
    else
        putstr("Failed to build image control...\n");
        while(CheckTouchExit() == 0);
// just hang if we didnt get the image list
        r := FAIL;
        goto quit;
    endif

    print ("Loaded ", img[IMG_COUNT], " images\n");
    print ("\nTouch and hold to exit...\n");
    pause(2000);

    pause(3000);
    gfx_Cls();

    repeat
        n := 0;

        while(n < img[IMG_COUNT] && !exit) // go through all images

            CheckTouchExit();           // if there's a press, exit

            img_SetPosition( img, n, (ABS(RAND() % 240)), (ABS(RAND() %
320))); // spread out the images

            n++;

        wend

        img_Show(img, ALL);           // update the entire control in 1 hit

    until(exit);

quit:

    mem_Free(img);           // release the image control

```



```
        file_Unmount();    // (program must release all resources)
        return r;
endfunc
//=====
```

2.23.11. spiflash_PlayWAV(spi#, Enablepin)

Syntax	<code>spiflash_PlayWAV(spi#, Enablepin) ;</code>	
Arguments	<code>spi#, Enablepin</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	<code>value</code>	
	value	<p>If there are no errors, returns number of blocks to play (1 to 32767)</p> <p>If errors occurred, the following is returned</p> <ul style="list-style-type: none"> -7 : Insufficient memory available for WAV buffer and file -6 : cant play this rate -5 : no data chunk found in first rsector -4 : no format data -3 : no wave chunk signature -2 : bad wave file format -1 : file not found
Description	<p>Play a wave file from the FLASH memory device on the specified SPI port and enable pin. The source is the address set by <code>spiflash_SetAdd()</code>, or incremented by subsequent reads or writes. Opens the wav file, decode the header to set the appropriate wave player parameters and set off the playing of the file as a background process.</p> <p>This function automatically grabs a chunk of memory for a wave buffer. The minimum memory requirement is the wave buffer size of 1024. The size of the wave buffer allocation can be increased by the <code>snd_BufSize</code> function.</p> <p>The default size 1024 bytes.</p> <p>Note: The memory is only required during the duration of play, and is automatically released while not in use.</p> <p>See "Sound Control Functions" for additional play control functions.</p>	
Example	<pre>print("\nding.wav\n"); for(n:=0; n<45; n++) pitch := NOTES[n]; print([UDEC] pitch, "\r"); snd_Pitch(pitch); spiflash_PlayWAV(SPI1, PA0); while(snd_Playing()); //pause(500); next</pre>	

2.23.12. spiflash_PutC(char, spi#, Enablepin)

Syntax	<code>spiflash_PutC(char, spi#, Enablepin) ;</code>	
Arguments	char, spi#, Enablepin	
	char	Data byte about to be written.
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Nothing	
Description	This function writes the byte specified by "char" to the FLASH memory device on the specified SPI port and enable pin, at the position <code>spiflash_SetAdd()</code> , or incremented by subsequent reads or writes and advances the pointer appropriately (incremented by 1).	
Example	<code>spiflash_PutC('A', SPI1, PA0);</code>	

2.23.13. spiflash_PutS(source, spi#, Enablepin)

Syntax	<code>spiflash_PutS(source, spi#, Enablepin);</code>	
Arguments	source, spi#, Enablepin	
	source	A pointer to the string to be written.
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	count	
	count	Returns the number of characters written (excluding the null terminator).
Description	This function writes an ASCIIZ (null terminated) string from a buffer specified by " *source " to the FLASH memory device on the specified SPI port and enable pin, at the position set by spiflash_SetAdd() , or incremented by subsequent reads or writes and advances the pointer appropriately.	
Example	<code>spiflash_PutS(mystring, SPI1, PA0);</code>	

2.23.14. spiflash_PutW(word, spi#, Enablepin)

Syntax	<code>spiflash_PutW(word, spi#, Enablepin) ;</code>	
Arguments	word, spi#, Enablepin	
	word	Data about to be written
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Nothing	
Description	This function writes word sized (2 bytes) data specified by " word " to the FLASH memory device on the specified SPI port and enable pin, at the position indicated by set by <code>spiflash_SetAdd()</code> , or incremented by subsequent reads or writes and advances the pointer appropriately (incremented by 2).	
Example	<code>spiflash_PutW(0x1234, SPI1, PA0);</code>	

2.23.15. `spiflash_Read(destination, size, spi#, Enablepin)`

Syntax	<code>spiflash_Read(destination, size, spi#, Enablepin) ;</code>	
Arguments	destination, size, spi#, Enablepin	
	destination	Destination memory buffer
	size	Number of bytes to be read
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	count	
	count	Returns the number of characters read.
Description	<p>Reads the number of bytes specified by "size" from the FLASH memory device on the specified SPI port and enable pin into a destination memory buffer. The source is the address set by <code>spiflash_SetAdd()</code>, or incremented by subsequent reads or writes.</p> <p>If "destination" is zero, data is read direct to GRAM window</p>	
Example	<code>res := spiflash_Read(memblock, 20, SPI1, PA0);</code>	

2.23.16. spiflash_Run(spi#, Enablepin, arglistptr)

Syntax	<code>spiflash_Run(spi#, Enablepin, arglistptr) ;</code>	
Arguments	<code>spi#, Enablepin, arglistptr</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
	arglistptr	pointer to the list of arguments to pass to the new program.
Returns	Value	
	Value	Returns the value from main in the called program.
Description	<p>Any memory allocations in the main FLASH program are released, however, the stack and globals are maintained.</p> <p>If arglistptr is 0, no arguments are passed, else arglistptr points to an array, the first element being the number of additional elements in the array which contain the arguments.</p> <p>func 'main' in the called program accepts the arguments, if any.</p> <p>The arguments can only be passed by value, no pointers or references can be used as all memory is cleared before the file is loaded. Refer to spiflash_Exec and spiflash_LoadFunction for functions that can pass by reference.</p> <p>spiflash_SetAdd should have previously been called to identify the address of the program to be called.</p>	
Example	Refer to the <code>file_Run</code> example.	

2.23.17. spiflash_SetAdd(spi#, HiWord, LoWord)

Syntax	<code>spiflash_SetAdd(spi#, HiWord, LoWord) ;</code>	
Arguments	Hiword, LOword	
	Hiword	Specifies the high word (upper 2 bytes) of a 4 byte SPI FLASH memory byte address location.
	LOword	Specifies the low word (lower 2 bytes) of a 4 byte SPI FLASH memory byte address location.
	The arguments can be a variable, array element, expression or constant	
Returns	nothing	
Description	Set media memory internal Address pointer for to SPI FLASH memory.	
Example	<pre>spiflash_SetAdd(0, 513);</pre> <p>This example sets the SPI FLASH address to byte 513 for subsequent operations.</p>	

2.23.18. spiflash_SIG(spi#, Enablepin)

Syntax	<code>spiflash_SIG(spi#, Enablepin) ;</code>	
Arguments	<code>spi#, Enablepin</code>	
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Signature	
	Signature	Returns the Electronic Signature of the SPI FLASH device.
Description	Returns the Electronic Signature of the SPI FLASH device. Only the low order byte is valid, the upper byte is ignored.	
Example	<code>Sig := spiflash_SIG(SPI1, PA0) ;</code>	

2.23.19. spiflash_Write(Source, size, spi#, Enablepin)

Syntax	<code>spiflash_Write(Source, size, spi#, Enablepin) ;</code>	
Arguments	Source, size, spi#, Enablepin	
	source	Source memory buffer.
	size	Number of bytes to be written.
	Spi#	The SPI port to use, eg SPI1, SPI2 or SPI3.
	Enablepin	The enable pin assigned to this spiflash device, eg PA0
Returns	Status	
	Status	Returns TRUE if the Source address is valid
Description	Writes the number of bytes specified by "size" from the source buffer into the FLASH memory device on the specified SPI port and enable pin.	
Example	<code>res := spiflash_Write(memblock, 20, SPI1, PA0);</code>	

2.24. CRC Functions

Summary of Functions in this section:

- `crc_16(buf, count)`
- `crc_CCITT(buf, count, seed)`
- `crc_CSUM_8(buf, count)`
- `crc_MODBUS(buf, count)`

The CRC functions are mainly designed for serial communications, but are implemented in such a way that they can be used to other things as well.

The `com_TXblock` and `com_RXblock` commands can be used to assist with reading and writing comm ports, generating and checking CRCs with the minimum of user data manipulation.

2.24.1. `crc_16(buf, count)`

Syntax	<code>crc_16(buf, count) ;</code>	
Arguments	<code>buf, count</code>	
	<code>buf</code>	Source memory buffer. This is a string pointer.
	<code>count</code>	Number of bytes to be used to generate the CRC.
Returns	<code>CRC</code>	
	<code>CRC</code>	Returns the generated 16 bit CRC.
Description	<p>Calculates the Checksum CRC using the 'standard' 16 bit CRC algorithm.</p> <p>For the standard test string "123456789", <code>crc_16</code> will return 0xBB3D.</p> <p>Note if you calculate all of the incoming data INCLUDING the CRC, the result should be 0x00</p>	
Example	<code>Crc := crc_16(str_Ptr(buf), 10);</code>	

2.24.2. `crc_CCITT(buf, count, seed)`

Syntax	<code>crc_CCITT(buf, count, seed) ;</code>	
Arguments	buf, count, seed	
	buf	Source memory buffer. This is a string pointer.
	count	Number of bytes to be used to generate the CRC.
	seed	The seed for the CRC generation.
Returns	CRC	
	CRC	Returns the generated CCITT CRC.
Description	<p>Calculates the Checksum CRC as a 'standard' CRCITT checksum.</p> <p>For the standard test string "123456789", <code>crc_CCITT</code> with seed = 0 (XMODEM protocol) will return = 0x31C3, for seed = 0xFFFF, the result will be 0x29B1 and for seed = 0x1D0F, the result is 0xE5CC.</p> <p>Note if you calculate all of the incoming data INCLUDING the CRC, the result should be 0x00</p>	
Example	<code>Crc := crc_CCITT(str_Ptr(buf), 10, 0x0000);</code>	

2.24.3. crc_CSUM_8(buf, count)

Syntax	<code>crc_CSUM_8(buf, count) ;</code>	
Arguments	buf, count	
	buf	Source memory buffer. This is a string pointer.
	count	Number of bytes to be used to generate the CRC.
Returns	CRC	
	CRC	Returns the generated 8 bit checksum CRC.
Description	<p>Calculates the Checksum CRC as an 8 bit number. This is equivalent to simple addition of all bytes and returning the negated sum an 8 bit value.</p> <p>For the standard test string "123456789", <code>crc_CSUM_8</code> will return 0x0023.</p> <p>Note if you calculate all of the incoming data INCLUDING the CRC, the result should be 0x00</p>	
Example	<code>Crc := crc_CSUM_8(str_Ptr(buf), 10);</code>	

2.24.4. crc_MODBUS(buf, count)

Syntax	<code>crc_MODBUS(buf, count) ;</code>	
Arguments	buf, count	
	buf	Source memory buffer. This is a string pointer.
	count	Number of bytes to be used to generate the CRC.
Returns	CRC	
	CRC	Returns the generated MODBUS CRC.
Description	<p>Calculates the Checksum CRC as per the MODBUS standard.</p> <p>For the standard test string "123456789", crc_MODBUS will return 0x4B37.</p> <p>Note if you calculate all of the incoming data INCLUDING the CRC, the result should be 0x00</p>	
Example	<code>Crc := crc_MODBUS(str_Ptr(buf), 10);</code>	

3. System Registers Memory Map

The following tables outline in detail the DIABLO16 system registers and flags.

LABEL	ADDRESS		USAGE
	DEC	HEX	
RANDOM_LO	32	0x20	random number generator LO word
RANDOM_HI	33	0x21	random number generator HI word
SYSTEM_TIMER_LO	34	0x22	1msec 32 bit free running timer LO word
SYSTEM_TIMER_HI	35	0x23	1msec 32 bit free running timer HI word
TIMER0	36	0x24	1msec user timer 0
TIMER1	37	0x25	1msec user timer 1
TIMER2	38	0x26	1msec user timer 2
TIMER3	39	0x27	1msec user timer 3
TIMER4	40	0x28	1msec user timer 4
TIMER5	41	0x29	1msec user timer 5
TIMER6	42	0x2A	1msec user timer 6
TIMER7	43	0x2B	1msec user timer 7
SYS_X_MAX	44	0x2C	display hardware X res-1
SYS_Y_MAX	45	0x2D	display hardware Y res-1
GFX_XMAX	46	0x2E	current display width-1 determined by portrait / landscape swapping
GFX_YMAX	47	0x2F	current display height-1 determined by portrait / landscape swapping
GFX_LEFT	48	0x30	virtual left point for most recent object
GFX_TOP	49	0x31	virtual top point for most recent object
GFX_RIGHT	50	0x32	virtual right point for most recent object
GFX_BOTTOM	51	0x33	virtual bottom point for most recent object
GFX_X1	52	0x34	clipped left point for current object
GFX_Y1	53	0x35	clipped top point for current object
GFX_X2	54	0x36	clipped right point for current object
GFX_Y2	55	0x37	clipped bottom point for current object
GFX_X_ORG	56	0x38	current X origin
GFX_Y_ORG	57	0x39	current Y origin
GFX_THUMB_PERCENT	75	0x4B	size of slider thumb as percentage
GFX_THUMB_BORDER_DARK	76	0x4C	darker shadow of thumb
GFX_THUMB_BORDER_LIGHT	77	0x4D	lighter shadow of thumb
TOUCH_XMINCAL	78	0x4E	touch calibration value
TOUCH_YMINCAL	79	0x4F	touch calibration value
TOUCH_XMAXCAL	80	0x50	touch calibration value
TOUCH_YMAXCAL	81	0x51	touch calibration value
IMG_WIDTH	82	0x52	width of currently loaded image
IMG_HEIGHT	83	0x53	height of currently loaded image
IMG_FRAME_DELAY	84	0x54	if image, else inter frame delay for movie
IMG_FLAGS	85	0x55	bit 4 determines colour mode, other bits reserved
IMG_FRAME_COUNT	86	0x56	count of frames in a movie
IMG_PIXEL_COUNT_LO	87	0x57	count of pixels in the current frame
IMG_PIXEL_COUNT_HI	88	0x58	count of pixels in the current frame
IMG_CURRENT_FRAME	89	0x59	last frame shown
MEDIA_ADDRESS_LO	90	0x5A	micro-SD byte address LO
MEDIA_ADDRESS_HI	91	0x5B	micro-SD byte address HI
MEDIA_SECTOR_LO	92	0x5C	micro-SD sector address LO

NOTE: These registers are accessible with [peekW](#) and [pokeW](#) functions.

LABEL	ADDRESS		USAGE
	DEC	HEX	
MEDIA_SECTOR_HI	93	0x5D	micro-SD sector address HI
MEDIA_SECTOR_COUNT	94	0x5E	micro-SD number of bytes remaining in sector
TEXT_XPOS	95	0x5F	text current x pixel position
TEXT_YPOS	96	0x60	text current y pixel position
TEXT_MARGIN	97	0x61	text left pixel pos for carriage return
TXT_FONT_ID	98	0x62	font type, 0 = system font, else pointer to user font
TXT_FONT_MAX	99	0x63	max number of chars in font
TXT_FONT_OFFSET	100	0x64	starting offset (normally 0x20)
TXT_FONT_WIDTH	101	0x65	current font width
TXT_FONT_HEIGHT	102	0x66	Current font height
GFX_TOUCH_REGION_X1	103	0x67	touch capture region
GFX_TOUCH_REGION_Y	104	0x68	
GFX_TOUCH_REGION_X2	105	0x69	
GFX_TOUCH_REGION_Y2	106	0x6A	
GFX_CLIP_LEFT_VAL	107	0x6B	left clipping point (set with gfx_ClipWindow(...))
GFX_CLIP_TOP_VAL	108	0x6C	top clipping point (set with gfx_ClipWindow(...))
GFX_CLIP_RIGHT_VAL	109	0x6D	right clipping point (set with gfx_ClipWindow(...))
GFX_CLIP_BOTTOM_VAL	110	0x6E	bottom clipping point (set with gfx_ClipWindow(...))
GFX_CLIP_LEFT	111	0x6F	current clip value (reads full size if clipping turned off)
GFX_CLIP_TOP	112	0x70	current clip value (reads full size if clipping turned off)
GFX_CLIP_RIGHT	113	0x71	current clip value (reads full size if clipping turned off)
GFX_CLIP_BOTTOM	114	0x72	current clip value (reads full size if clipping turned off)
GRAM_PIXEL_COUNT_LO	115	0x73	LO word of count of pixels in the set GRAM area
GRAM_PIXEL_COUNT_HI	116	0x74	HI word of count of pixels in the set GRAM area
TOUCH_RAW_X	117	0x75	12 bit raw A2D X value from touch screen
TOUCH_RAW_Y	118	0x76	12 bit raw A2D Y value from touch screen
GFX_LAST_CHAR_WIDTH	119	0x77	calculated char width from last call to charWidth function
GFX_LAST_CHAR_HEIGHT	120	0x78	calculated height from last call to charHeight function
GFX_LAST_STR_WIDTH	121	0x79	calculated width from last call to strWidth function
GFX_LAST_STR_HEIGHT	122	0x7A	calculated height from last call to strHeight function
PIN_COUNTER_PA4	123	0x7B	pin counter for PA4
PIN_COUNTER_PA5	124	0x7C	pin counter for PA5
PIN_COUNTER_PA6	125	0x7D	pin counter for PA6
PIN_COUNTER_PA7	126	0x7E	pin counter for PA7
PIN_COUNTER_PA8	127	0x7F	pin counter for PA8
PIN_COUNTER_PA9	128	0x80	pin counter for PA9
PIN_EVENT_PA4	129	0x81	pin counter rollover event for PA4
PIN_EVENT_PA5	130	0x82	pin counter rollover event for PA5
PIN_EVENT_PA6	131	0x83	pin counter rollover event for PA6
PIN_EVENT_PA7	132	0x84	pin counter rollover event for PA7
PIN_EVENT_PA8	133	0x85	pin counter rollover event for PA8
PIN_EVENT_PA9	134	0x86	pin counter rollover event for PA9
QEN1_COUNTER_LO	135	0x87	quadrature encoder #1 counter LO
QEN1_COUNTER_HI	136	0x88	quadrature encoder #1 counter HI
QEN1_DELTA	137	0x89	quadrature encoder #1 delta count
QEN2_COUNTER_LO	138	0x8A	quadrature encoder #2 counter LO
QEN2_COUNTER_HI	139	0x8B	quadrature encoder #2 counter HI
QEN2_DELTA	140	0x8C	quadrature encoder #2 delta count
FALSE_REASON	141	0x8D	explanation 'false' results, currently only for flash_ functions

NOTE: These registers are accessible with [peekW](#) and [pokeW](#) functions.

4. Appendix A : Runtime Error Messages

Error No.	Error Meaning	Notes
1	Failed to receive 'L' during loading process from Workshop	Not in DIABLO16
2	Did not receive valid header info from Workshop	Unexpected error during Program load
3	Header size does not match loader info	Not in DIABLO16
4	Could not allocate enough memory for program	Unexpected error during Program load
5	Loader checksum error	Unexpected error during Program load
6	Did not receive header prior to 'L' command	Not in DIABLO16
7	Header size entry does not match loader value	Unexpected error during Program load
8	Failed to load program from FLASH	Internal
9	Could not allocate code segment	Not in DIABLO16
10	Could not load function file from disk	File on disk possibly corrupted
11	Bad header in program file	File on disk possibly corrupted
12	Header in program file differs from file size	File on disk possibly corrupted
13	Could not allocate global memory for program file	Program probably too large
14	Program File checksum error	File on disk possibly corrupted
15	EVE Stack Overflow	Infinitely recursive program or insufficient Stack Size
16	Unsupported PmmC function	Program error, or .fnc file mismatch
17	Illegal COM0 Event Function address	Program error
18	Illegal COM1, COM2, or COM3 Event Function address	Program error
19	Bad txt_Set(...) command number	Program error
20	Bad gfx_Get(...) command number	Program error
21	Bad gfx_Set(...) command number	Program error
22	Bad address for peekW or pokeW	Program error
23	Bad timer number for Timer function	Program error
24	Bad Event for sys_SetTimerEvent(...)	Program error
25	Flash Write Verify Failed	Internal
26	Bad or missing uSD Card	Program specifies #MODE of 'save to disk', but no valid disk can be found
27	Illegal Event Function Address	Program error
28	Not a pre-defined baud rate	Program error in setbaud()
29	Target of flash_Exec cannot have globals or privates	Program error

5. Hardware Tools

The following hardware tools are required for full control of the DIABLO16 Processor.

5.1. 4D Programming Cable & Adaptor

The 4D Programming Cable and uUSB-PA5 Programming Adaptors are essential hardware tools to program, customise and test the DIABLO16 Processor.

Either the 4D Programming Cable or the uUSB-PA5 Programming Adaptor can be used.

The 4D programming interfaces are used to program a new Firmware/PmmC, Display Driver and for downloading compiled 4DGL code into the processor. They even serve as an interface for communicating serial data to the PC.

The 4D Programming Cable and uUSB-PA5 Programming Adaptor are available from 4D Systems, www.4dsystems.com.au

Using a non-4D programming interface could damage your processor, and **void your Warranty.**



4D Programming Cable



uUSB-PA5 Programming Adaptor

5.2. Evaluation Display Modules

The following modules, available from 4D Systems, can be used for evaluation purposes to discover what the DIABLO16 processor has to offer.



uLCD-70DT – 7.0" Intelligent DIABLO16 Display

Other modules are also available. Please contact 4D Systems for more information, or visit the 4D Systems website, www.4dsystems.com.au

5.3. Memory Cards - FAT16 Format

The DIABLO16 Processor uses off the shelf standard SDHC/SD/micro-SD memory cards with up to 4GB capacity usable with FAT16 formatting. For any FAT file related operations, before the memory card can be used it must first be formatted with FAT16 option. The formatting of the card can be done on any PC system with a card reader. Select the appropriate drive and choose the FAT16 (or just FAT in some systems) option when formatting. The card is now ready to be used in the DIABLO16 based application.



The DIABLO16 Processor also supports high capacity HC memory cards (4GB and above). The available capacity of SD-HC cards varies according to the way the card is partitioned and the commands used to access it.

The FAT partition is always first (if it exists) and can be up to the maximum size permitted by FAT16. Windows 7 will format FAT16 up to 4GB. Windows XP will format FAT16 up to 2GB and the Windows XP command prompt will format FAT16 up to 4GB. Please refer to the RMPET tool in Workshop4.

Note: A microSD card capable of SPI is a requirement for all 4D Systems' display modules powered by Goldelox, Picaso or Diablo16 Processors. If a non-SPI compatible card is used, it will simply fail to mount, or may cause intermittent issues resulting in lock ups and crashing of the application. Please refer to the 4D Systems website for microSD cards offered by 4D Systems.

6. 4D Systems - Workshop 4 IDE

Workshop4 is a comprehensive software IDE that provides an integrated software development platform for all of the 4D family of processors and modules. The IDE combines the Editor, Compiler, Linker and Downloader to develop complete 4DGL application code. All user application code is developed within the Workshop 4 IDE.



The Workshop 4 IDE supports multiple development environments for the user, to cater for different user requirements and skill level.

- The **Designer** environment enables the user to write 4DGL code in its natural form to program the 4D processor of choice.
- A visual programming experience, suitably called **ViSi**, enables drag-and-drop type placement of objects to assist with 4DGL code generation and allows the user to visualise how the display will look while being developed.
- An advanced environment called **ViSi-Genie** doesn't require any 4DGL coding at all, it is all done automatically for you. Simply lay the display out with the objects you want, set the events to drive them and the code is written for you automatically. ViSi-Genie provides the latest rapid development experience from 4D Systems.

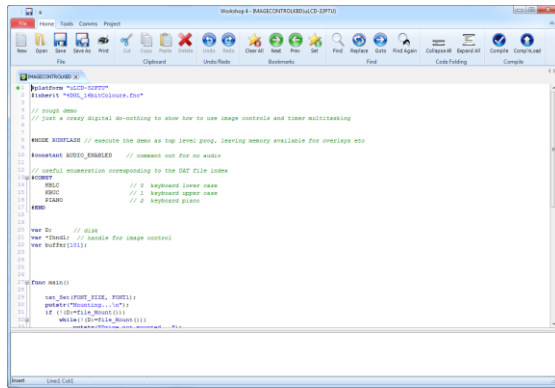
The Workshop 4 IDE is available from the 4D Systems website. www.4dsystems.com.au

For a comprehensive manual on the Workshop 4 IDE Software along with other documents, refer to the documentation from the 4D Systems website, on the Workshop 4 product page.

6.1. Workshop 4 – Designer Environment

Choose the Designer environment to write 4DGL code in its raw form.

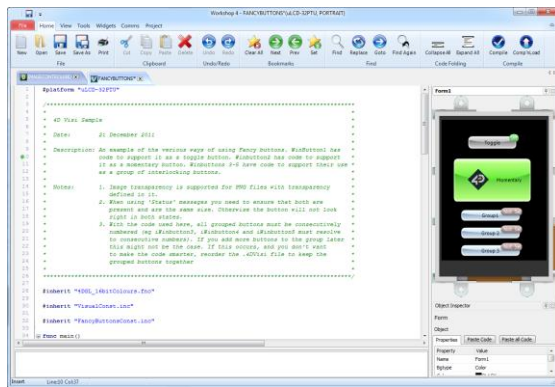
The Designer environment provides the user with a simple yet effective programming environment where pure 4DGL code can be written, compiled and downloaded to the DIABLO16.



6.2. Workshop 4 – ViSi Environment

ViSi was designed to make the creation of graphical displays a more visual experience.

ViSi is a great software tool that allows the user to see the instant results of their desired graphical layout. Additionally, there is a selection of inbuilt dials, gauges and meters that can simply be placed onto the simulated module display. From here each object can have its properties edited, and at the click of a button all relevant 4DGL code associated with that object is produced in the user program. The user can then write 4DGL code around these objects to utilise them in the way they choose.



6.3. Workshop 4 – ViSi Genie Environment

ViSi Genie is a breakthrough in the way 4D Systems' graphic display modules are programmed. It is an environment like no other, a code-less programming environment that provides the user with a rapid visual experience, enabling a simple GUI application to be 'written' from scratch in literally seconds.

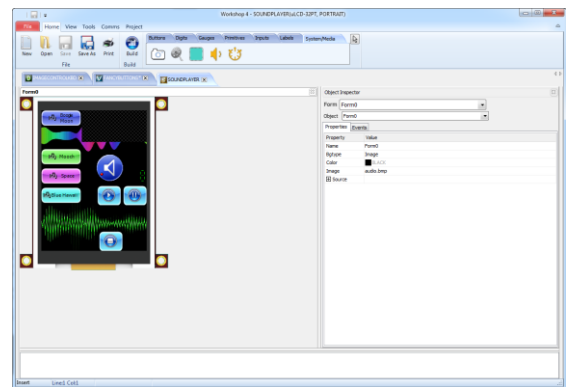
ViSi Genie does all the background coding, no 4DGL to learn, it does it all for you.

Pick and choose the relevant objects to place on the display, much like the ViSi Environment, yet without having to write a single line of code. Each object has parameters which can be set, and configurable events to animate and drive other objects or communicate with external devices.

Simply place an object on the screen, position and size it to suit, set the parameters such as colour, range, text, and finally select the event you wish the object to be associated with, it is that simple.

In seconds you can transform a blank display into a fully animated GUI with moving sliders, animated press and release buttons, and much more. All without writing a single line of code!

ViSi Genie provides the user with a feature rich rapid development environment, second to none.



6.4. Workshop 4 – Serial Environment

The Serial environment in the Workshop 4 IDE provides the user the ability to transform the Diablo16 into a slave serial graphics controller.

This enables the user to use their favourite microcontroller or serial device as the Host, without having to learn 4DGL or program in a separate IDE. Once the Picaso is configured and downloaded to from the Serial Environment, simple graphic commands can be sent from the users host microcontroller to display primitives, images, sound or even video.

Refer to the “**Serial Command Set Reference Manual**” from the Workshop 4 product page on the 4D Systems website for a complete listing of all the supported serial commands

By default, each module shipped from the 4D Systems factory will come pre-programmed ready for use in the Serial mode, unless it comes as part of a Starter Kit.

7. Revision History

Revision History		
Revision	Revision Content	Revision Date
1.0	First Release	22/07/2013
1.1	Added new Functions disp_Disconnect(), disp_Init() and sys_DeepSleep(). Fix spelling mistake in file_LoadImageControl	22/10/2013
1.2	Fixed gfx_Contrast description as it contained Picaso information, other minor non-functional related fixes.	07/11/2013
1.3	Added com_TXblock, com1_TXblock, com2_TXblock and com3_TXblock, com_RXblock, com1_RXblock, com2_RXblock and com3_RXblock, com_Mode, crc_CSUM_8, crc_16, crc_MODBUS and crc_CCITT for support of CRCs, non 8N1 mode and block transmit and receive Added spiflash_SIG, spiflash_ID, spiflash_BulkErase, spiflash_BlockErase, spiflash_SetAdd, spiflash_Read, spiflash_Write spiflash_Image, spiflash_PutC, spiflash_GetC, spiflash_PutW, spiflash_GetW, spiflash_PutS, spiflash_GetS spiflash_LoadFunction, spiflash_Run, spiflash_Exec, spiflash_LoadImageControl, spiflash_PlayWAV for support if SPI Flash memory Fixed error return codes in file_PlayWAV and added missing code.	23/12/2013
1.4	Added bus_Read8 and bus_Write8	06/01/2014
1.5	Added Mode PWM_BINARY and usage notes. Added notes to spiflash initialization. Added disp_BlitPixelsFromCOMx. Added special baud rates to com_SetBaud(). Added spix_ReadBlock and spix_Writeblock. All these additions apply to PmmC 1.1 and later.	25/02/2014
1.6	Documented v1.1 PmmC's changes to files opened in append mode. Added new I2C options.	21/03/2014
1.7	Documented V1.3 PmmC's new snd_Freq(), sys_GetDateVar(), sys_GetTimeVar() and pin_PulseoutCount() functions.	07/07/2014
1.8	Added keywords Backlight and Brightness to assist searchers finding the contrast setting. Fixed format of date in sys_GetDate function.	04/08/2014
1.9	Documented V1.5 PmmC's new txt_FontBank, putnumXY, flt_PRINTxy, file_Rename, file_Setdate, NP_Write and OW_* functions. Fixed error in memory size used for file_Mount. Added detail to set_Clippling().	16/09/2014
1.10	Updated information for file_LoadImageControl mode 2. Updated control block size in file_Mount. Added information about source of uSD based font in txt_FontID. Added information about the use of TRANSPARENCY. Fixed spelling of snd_Freq in example. Added information about the SPIx_Write and SPIx_Read operations. Clarified information about events.	22/12/2014
1.11	Added more information about interrupts and NP_Write. Added notes to comx_TXbufferHold. Fixed case of pwm_Init()	11/02/2015
1.12	Modification to Analog Input read rates	06/03/2015
1.13	Fixed I2Cx_Write return code information. Clarified str_Length and bus_SetChangeInterrupt examples. Fixed syntax example for usub_3232. Added 'page' options to gfx_Set for uLCD-43D* displays.	07/05/2015
1.14	Fixed FontIDs for deja fonts. Updated udiv_3232 sample. Improved return description for str_Match and str_MatchI. Added str_Printf to 'to' function. Added runtime Error 29. Improved examples for str_Cat, str_CatN, str_Find, str_FindI, str_Match, str_MatchI and file_Exec.	14/07/2015
1.15	Corrected flt_POW syntax typo. Corrected incurred information relating to PA14 and PA15. Improvements to a few pin_ and bus_ function examples.	06/10/2015

8. Legal Notice

Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems. 4D Systems reserves the right to modify, update or make changes to Specifications or written material without prior notice at any time.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.

9. Contact Information

For Technical Support: www.4dsystems.com.au/support

For Sales Support: sales@4dsystems.com.au

Website: www.4dsystems.com.au

Copyright 4D Systems Pty. Ltd. 2000-2015.