



PRODUCT MANUAL

# Dynamic C<sup>®</sup>

Integrated C Development System  
For Rabbit Microprocessors

## Function Reference Manual

019-0113 • 080702-L

The latest revision of this manual is available on the Rabbit web site,  
[www.rabbit.com](http://www.rabbit.com), for free, unregistered download.

---

# Dynamic C Function Reference Manual

Part Number 019-0113 • 080702-L • Printed in U.S.A.

Digi International Inc. © 2007-2008 • All rights reserved.

No part of the contents of this manual may be reproduced or transmitted in any form or by any means without the express written permission of Digi International Inc.

Permission is granted to make one or more copies as long as the copyright page contained therein is included. These copies of the manuals may not be let or sold for any reason without the express written permission of Digi International Inc.

Digi International Inc reserves the right to make changes and improvements to its products without providing notice.

## Trademarks

RabbitSys™ is a trademark of Digi International Inc.

Rabbit and Dynamic C® are registered trademarks of Digi International Inc.

Windows® is a registered trademark of Microsoft Corporation

# Table of Contents

<b>Alphabetical Listing of Dynamic C Functions</b>	<b>5</b>
<b>Group Listing of Dynamic C Functions</b>	<b>15</b>
Arithmetic	15
Bit Manipulation	15
Bus Operation (Rabbit 3000, 4000)	15
Character	15
Data Encryption	15
Direct Memory Access (Rabbit 4000, 5000)	15
Dynamic Memory Allocation	16
ECC	16
Error Handling	16
Extended Memory	17
Fast Fourier Transforms	17
File Compression	17
File System, FAT	17
File System, FS1	18
File System, FS2	18
Flash, NAND	18
Flash, Parallel	18
Flash, SD	18
Flash, Serial	19
Floating-Point Math	19
Global Positioning System	19
HDLC Protocol (Rabbit 3000, 4000, 5000)	19
I/O	20
I2C Protocol	20
Interrupts	20
Logging Subsystem	20
MD5	20
MicroC/OS-II	20
Miscellaneous	21
Multitasking	21
Number-to-String Conversion	22
Partitions	22
Pulse Width Modulation (Rabbit 3000, 4000, 5000)	22
Quadrature Decoder (Rabbit 3000, 4000, 5000)	22
Rabbit 3000, 4000	22
Rabbit 3000, 4000, 5000	22
Rabbit 4000, 5000	23
Real-Time Clock	23
Serial Communication	24
Serial Packet Driver	26
Servo Control (Rabbit 3000, 4000)	27
SPI	27
Stdio	27
String Manipulation	27
String-to-Number Conversion	28
System	28
User Block	28
VBAT RAM (Rabbit 4000, 5000)	28
Watchdogs	28
<b>Chapter 1: Function Descriptions</b>	<b>29</b>
<b>Software License Agreement</b>	<b>563</b>



## Alphabetical Listing of Dynamic C Functions

New releases of Dynamic C often contain new API functions. You can check if your version of Dynamic C contains a particular function by checking the Function Lookup feature in the Help menu. If you see functions described in this manual that you want but do not have, please consider updating your version of Dynamic C. To update Dynamic C, go to: [www.rabbit.com/products/dc/](http://www.rabbit.com/products/dc/) or call 1.530.757.8400.

### Symbols

<code>_GetSysMacroIndex</code>	202
<code>_GetSysMacroValue</code>	203
<code>_sysIsSoftReset</code>	525
<code>_xalloc</code>	550
<code>_xavail</code>	553

### A

<code>abs</code>	32
<code>acos</code>	32
<code>acot</code>	33
<code>acsc</code>	33
<code>AESdecrypt</code>	34
<code>AESdecryptStream</code>	34
<code>AESencrypt</code>	35
<code>AESencryptStream</code>	36
<code>AESexpandKey</code>	37
<code>AESinitStream</code>	38
<code>asec</code>	39
<code>asin</code>	39
<code>atan</code>	40
<code>atan2</code>	41
<code>atof</code>	42
<code>atoi</code>	43
<code>atol</code>	44

### B

<code>BIT</code>	46
<code>bit</code>	45
<code>BitRdPortE</code>	47
<code>BitRdPortI</code>	48
<code>BitWrPortE</code>	49
<code>BitWrPortI</code>	50

### C

<code>CalculateECC256</code>	51
<code>ceil</code>	53
<code>ChkCorrectECC256</code>	52
<code>chkHardReset</code>	54

<code>chkSoftReset</code>	54
<code>chkWDTO</code>	55
<code>clockDoublerOff</code>	56
<code>clockDoublerOn</code>	56
<code>CloseInputCompressedFile</code>	57
<code>CloseOutputCompressedFile</code>	57
<code>CoBegin</code>	58
<code>cof_pktAreceive</code>	58
<code>cof_pktAsend</code>	59
<code>cof_pktBreceive</code>	58
<code>cof_pktBsend</code>	59
<code>cof_pktCreceive</code>	58
<code>cof_pktCsend</code>	59
<code>cof_pktDreceive</code>	58
<code>cof_pktDsend</code>	59
<code>cof_pktEreceive</code>	58
<code>cof_pktEsend</code>	59
<code>cof_pktFreceive</code>	58
<code>cof_pktFsend</code>	59
<code>cof_serAgetc</code>	60
<code>cof_serAgets</code>	61
<code>cof_serAputc</code>	62
<code>cof_serAputs</code>	63
<code>cof_serAread</code>	64
<code>cof_serAwrite</code>	65
<code>cof_serBgetc</code>	60
<code>cof_serBgets</code>	61
<code>cof_serBputc</code>	62
<code>cof_serBputs</code>	63
<code>cof_serBread</code>	64
<code>cof_serBwrite</code>	65
<code>cof_serCgetc</code>	60
<code>cof_serCgets</code>	61
<code>cof_serCputc</code>	62
<code>cof_serCputs</code>	63
<code>cof_serCread</code>	64
<code>cof_serCwrite</code>	65
<code>cof_serDgetc</code>	60
<code>cof_serDgets</code>	61
<code>cof_serDputc</code>	62
<code>cof_serDputs</code>	63

cof_serDread	64
cof_serDwrite	65
cof_serEgetc	60
cof_serEgets	61
cof_serEputc	62
cof_serEputs	63
cof_serEread	64
cof_serEwrite	65
cof_serFgetc	60
cof_serFgets	61
cof_serFputc	62
cof_serFputs	63
cof_serFread	64
cof_serFwrite	65
CompressFile	66
CoPause	67
CoReset	67
CoResume	68
cos	69
cosh	69

## D

DecompressFile	70
defineErrorHandler	71
deg	72
DelayMs	73
DelaySec	74
DelayTicks	74
Disable_HW_WDT	75
DMAalloc	76
DMAcompleted	77
DMAhandle2chan	78
DMAioe2mem	79
DMAioi2mem	81
DMAloadBufDesc	82
DMAmatchSetup	83
DMAmem2ioe	84
DMAmem2ioi	85
DMAmem2mem	86
DMApoll	87
DMAprintBufDesc	88
DMAprintRegs	89
DMAsetBufDesc	90
DMAsetDirect	91
DMAsetParameters	92
DMAstartAuto	93
DMAstartDirect	94
DMAstop	95
DMAstopDirect	96
DMAtimerSetup	96
DMAunalloc	97

## E

enableIObus	98
-------------	----

errlogFormatEntry	100
errlogFormatRegDump	101
errlogFormatStackDump	101
errlogGetHeaderInfo	99
errlogGetMessage	102
errlogGetNthEntry	100
errlogReadHeader	102
error_message	103
exception	104
exit	105
exp	105

## F

fabs	106
fat_AutoMount	107
fat_Close	110
fat_CreateDir	111
fat_CreateFile	112
fat_CreateTime	113
fat_Delete	114
fat_EnumDevice	115
fat_EnumPartition	116
fat_FileSize	117
fat_FormatDevice	118
fat_FormatPartition	119
fat_Free	120
fat_GetAttr	121
fat_GetName	122
fat_Init	123
fat_InitUCOSMutex	124
fat_LastAccess	125
fat_LastWrite	126
fat_MountPartition	127
fat_Open	128
fat_OpenDir	130
fat_PartitionDevice	131
fat_Read	132
fat_ReadDir	133
fat_Seek	135
fat_SetAttr	137
fat_Split	138
fat_Status	139
fat_SyncFile	140
fat_SyncPartition	141
fat_Tell	142
fat_tick	143
fat_Truncate	144
fat_UnmountDevice	145
fat_UnmountPartition	146
fat_Write	147
fat_xWrite	148
fclose	149
fcreate	150
fcreate (FS2)	151

fcreate_unused	152
fcreate_unused (FS2)	153
fdelete	154
fdelete (FS2)	155
fflush (FS2)	156
ftcplx	157
ftcplxinv	158
ftreal	159
ftrealinv	160
flash_erasechip	161
flash_erasector	162
flash_gettype	163
flash_init	164
flash_read	165
flash_readsector	166
flash_sector2xwindow	167
flash_writesector	168
floor	169
fmod	169
fopen_rd (FS1)	170
fopen_rd (FS2)	171
fopen_wr	172
fopen_wr (FS2)	173
forceSoftReset	174
fread	174
fread (FS2)	175
frexp	176
fs_format (FS1)	177
fs_format (FS2)	178
fs_get_flash_lx (FS2)	184
fs_get_lx (FS2)	185
fs_get_lx_size (FS2)	186
fs_get_other_lx (FS2)	187
fs_get_ram_lx (FS2)	188
fs_init (FS1)	179
fs_init (FS2)	180
fs_reserve_blocks (FS1)	181
fs_set_lx (FS2)	189
fs_setup (FS2)	190
fs_sync (FS2)	192
fsck	181
fseek (FS1)	182
fseek (FS2)	183
fshift	195
ftell (FS1)	193
ftell (FS2)	194
ftoa	199
fwrite (FS1)	197
fwrite (FS2)	198

## G

getchar	199
getcrc	200
getdivider19200	200

gets	201
GetVectExtern2000	204
GetVectExtern3000	205
GetVectIntern	206
gps_get_position	206
gps_get_utc	207
gps_ground_distance	207

## H

hanncplx	208
hannreal	209
HDLcAbortE	210
HDLcAbortF	210
HDLcCloseE	210
HDLcCloseF	210
HDLcDropE	211
HDLcDropF	211
HDLcErrorE	211
HDLcErrorF	211
HDLcExtClockE	212
HDLcExtClockF	212
HDLcOpenE	213
HDLcOpenF	213
HDLcPeekE	214
HDLcPeekF	214
HDLcReceiveE	215
HDLcReceiveF	215
HDLcSendE	216
HDLcSendF	216
HDLcSendingE	217
HDLcSendingF	217
hexstrtobyte	217
hitwd	218
htoa	218

## I

i2c_check_ack	230
i2c_init	231
i2c_read_char	231
i2c_send_ack	232
i2c_send_nak	232
i2c_start_tx	233
i2c_startw_tx	234
i2c_stop_tx	235
i2c_write_char	235
IntervalMs	219
IntervalSec	219
IntervalTick	220
ipres	220
ipset	221
isalnum	221
isalpha	222
isctrl	222
isCoDone	223

isCoRunning .....	223
isdigit .....	224
isgraph .....	225
islower .....	225
isprint .....	226
ispunct .....	227
isspace .....	226
isupper .....	228
isxdigit .....	228
itoa .....	229

## K

kbhit .....	236
-------------	-----

## L

labs .....	236
ldexp .....	237
log .....	237
log_clean .....	238
log_close .....	239
log_condition .....	240
log_format .....	241
log_map .....	242
log_next .....	243
log_open .....	244
log_prev .....	245
log_put .....	246
log_seek .....	247
log10 .....	248
longjmp .....	248
loophead .....	249
loopinit .....	249
lsqrt .....	250
ltoa .....	250
ltoan .....	251
lx_format .....	252

## M

mbr_CreatePartition .....	253
mbr_EnumDevice .....	254
mbr_FormatDevice .....	255
mbr_MountPartition .....	256
mbr_UnmountPartition .....	257
mbr_ValidatePartitions .....	258
md5_append .....	259
md5_finish .....	260
md5_ini .....	259
memchr .....	261
memcmp .....	262
memcpy .....	263
memmove .....	264
memset .....	265
mktime .....	266

mktm .....	267
modf .....	268

## N

nf_eraseBlock .....	269
nf_getPageCount .....	270
nf_getPageSize .....	271
nf_initDevice .....	272
nf_InitDriver .....	274
nf_isBusyRBHW .....	275
nf_isBusyStatus .....	276
nf_readPage .....	277
nf_writePage .....	278
nf_XD_Detect .....	279

## O

OpenInputCompressedFile .....	280
OpenOutputCompressedFile .....	281
OS_ENTER_CRITICAL .....	282
OS_EXIT_CRITICAL .....	282
OSFlagAccept .....	283
OSFlagCreate .....	285
OSFlagDel .....	286
OSFlagPend .....	287
OSFlagPost .....	289
OSFlagQuery .....	290
OSInit .....	291
OSMboxAccept .....	291
OSMboxCreate .....	292
OSMboxDel .....	293
OSMboxPend .....	294
OSMboxPost .....	295
OSMboxPostOpt .....	296
OSMboxQuery .....	297
OSMemCreate .....	298
OSMemGet .....	299
OSMemPut .....	300
OSMemQuery .....	301
OSMutexAccept .....	302
OSMutexCreate .....	303
OSMutexDel .....	304
OSMutexPend .....	305
OSMutexPost .....	306
OSMutexQuery .....	307
OSQAccept .....	308
OSQCreate .....	309
OSQDel .....	310
OSQFlush .....	311
OSQPend .....	312
OSQPost .....	313
OSQPostFront .....	314
OSQPostOpt .....	315
OSQQuery .....	316
OSSchedLock .....	317



OSSchedUnlock .....	317	pktAreceive .....	360
OSSemAccept .....	318	pktAsend .....	361
OSSemCreate .....	319	pktAsending .....	362
OSSemPend .....	319	pktAsetParity .....	362
OSSemPost .....	320	pktBclose .....	356
OSSemQuery .....	321	pktBgetErrors .....	356
OSSetTickPerSec .....	322	pktBinitBuffers .....	357
OSStart .....	322	pktBopen .....	358
OSStatInit .....	323	pktBreceive .....	360
OSTaskChangePrio .....	323	pktBsend .....	361
OSTaskCreate .....	324	pktBsending .....	362
OSTaskCreateExt .....	325	pktBsetParity .....	362
OSTaskCreateHook .....	326	pktBclose .....	356
OSTaskDel .....	327	pktCgetErrors .....	356
OSTaskDelHook .....	328	pktCinitBuffers .....	357
OSTaskDelReq .....	329	pktCopen .....	358
OSTaskIdleHook .....	330	pktCreceive .....	360
OSTaskQuery .....	330	pktCsend .....	361
OSTaskResume .....	331	pktCsending .....	362
OSTaskStatHook .....	331	pktCsetParity .....	362
OSTaskStkChk .....	332	pktDclose .....	356
OSTaskSuspend .....	333	pktDgetErrors .....	356
OSTaskSwHook .....	334	pktDinitBuffers .....	357
OSTCBInitHook .....	334	pktDopen .....	358
OSTimeDly .....	335	pktDreceive .....	360
OSTimeDlyHMSM .....	336	pktDsend .....	361
OSTimeDlyResume .....	337	pktDsending .....	362
OSTimeDlySec .....	338	pktDsetParity .....	362
OSTimeGet .....	339	pktEclose .....	356
OSTimeSet .....	339	pktEgetErrors .....	356
OSTimeTick .....	340	pktEinitBuffers .....	357
OSTimeTickHook .....	340	pktEopen .....	358
OSVersion .....	341	pktEreceive .....	360
outchrs .....	341	pktEsend .....	361
outstr .....	342	pktEsending .....	362
		pktEsetParity .....	362
		pktFclose .....	356
		pktFgetErrors .....	356
		pktFinitBuffers .....	357
		pktFopen .....	358
		pktFreceive .....	360
		pktFsend .....	361
		pktFsending .....	362
		pktFsetParity .....	362
		plast .....	363
		plast_fast .....	364
		pmovebetween .....	365
		pmovebetween_fast .....	367
		pnel .....	368
		pnext .....	369
		pnext_fast .....	370
		poly .....	371
		pool_append .....	372
		pool_init .....	373
<b>P</b>			
paddr .....	343		
paddrDS .....	344		
paddrSS .....	345		
palloc .....	346		
palloc_fast .....	347		
pavail .....	348		
pavail_fast .....	349		
pccalloc .....	350		
pfirst .....	351		
pfirst_fast .....	352		
pfree .....	353		
pfree_fast .....	354		
phwm .....	355		
pktAclose .....	356		
pktAgetErrors .....	356		
pktAinitBuffers .....	357		
pktAopen .....	358		

pool_link .....	374
pool_xappend .....	375
pool_xinit .....	376
pow .....	377
pow10 .....	377
powerspectrum .....	378
pprev .....	379
pprev_fast .....	380
pputlast .....	381
pputlast_fast .....	382
premain .....	382
preorder .....	383
printf .....	385
putchar .....	391
puts .....	391
pwm_init .....	392
pwm_set .....	393
pxalloc .....	394
pxalloc_fast .....	395
pxcalloc .....	396
pxfirst .....	397
pxfirst_fast .....	398
pxfree .....	399
pxfree_fast .....	400
pxlast .....	401
pxlast_fast .....	402
pxnext .....	403
pxnext_fast .....	404
pxprev .....	405
pxprev_fast .....	406

## Q

qd_error .....	407
qd_init .....	408
qd_read .....	409
qd_zero .....	409
qsort .....	410

## R

rad .....	411
rand .....	411
randb .....	412
randg .....	412
RdPortE .....	413
RdPortI .....	414
read_rtc .....	416
read_rtc_32kHz .....	416
ReadCompressedFile .....	415
readUserBlock .....	417
readUserBlockArray .....	418
RES .....	419
res .....	419
ResetErrorLog .....	420
root2vram .....	420

root2xmem .....	421
rtc_timezone .....	422
runwatch .....	423

## S

sdspi_debounce .....	423
sdspi_get_csd .....	424
sdspi_get_scr .....	425
sdspi_get_status_reg .....	426
sdspi_getSectorCount .....	426
sdspi_init_card .....	427
sdspi_initDevice .....	428
sdspi_isWriting .....	429
sdspi_notbusy .....	429
sdspi_print_dev .....	430
sdspi_process_command .....	431
sdspi_read_sector .....	432
sdspi_reset_card .....	433
sdspi_sendingAP .....	434
sdspi_set_block_length .....	435
sdspi_setLED .....	434
sdspi_write_sector .....	437
sdspi_WriteContinue .....	436
serAclose .....	456
serAdatabits .....	456
serAdmaOff .....	457
serAdmaOn .....	458
serAflowcontrolOff .....	459
serAflowcontrolOn .....	460
serAgetc .....	461
serAgetError .....	462
serAopen .....	463
serAparity .....	464
serApeek .....	465
serAputc .....	466
serAputs .....	467
serArdFlush .....	468
serArdFree .....	468
serArdUsed .....	469
serAread .....	470
serAwrFlush .....	471
serAwrFree .....	471
serAwrite .....	472
serAwrUsed .....	473
serBclose .....	456
serBdatabits .....	456
serBdmaOff .....	457
serBdmaOn .....	458
serBflowcontrolOff .....	459
serBflowcontrolOn .....	460
serBgetc .....	461
serBgetError .....	462
serBopen .....	463
serBparity .....	464

serBpeek .....	465	serEdatabits .....	456
serBputc .....	466	serEdmaOff .....	457
serBputs .....	467	serEdmaOn .....	458
serBrdFlush .....	468	serEflowcontrolOff .....	459
serBrdFree .....	468	serEflowcontrolOn .....	460
serBrdUsed .....	469	serEgetc .....	461
serBread .....	470	serEgetError .....	462
serBwrFlush .....	471	serEopen .....	463
serBwrFree .....	471	serEparity .....	464
serBwrite .....	472	serEpeek .....	465
serBwrUsed .....	473	serEputc .....	466
serCclose .....	456	serEputs .....	467
serCdatabits .....	456	serErdFlush .....	468
serCdmaOff .....	457	serErdFree .....	468
serCdmaOn .....	458	serErdUsed .....	469
serCflowcontrolOff .....	459	serEread .....	470
serCflowcontrolOn .....	460	serEwrFlush .....	471
serCgetc .....	461	serEwrFree .....	471
serCgetError .....	462	serEwrite .....	472
serCopen .....	463	serEwrUsed .....	473
serCparity .....	464	serFclose .....	456
serCpeek .....	465	serFdatabits .....	456
serCputc .....	466	serFdmaOff .....	457
serCputs .....	467	serFdmaOn .....	458
serCrdFlush .....	468	serFflowcontrolOff .....	459
serCrdFree .....	468	serFflowcontrolOn .....	460
serCrdUsed .....	469	serFgetc .....	461
serCread .....	470	serFgetError .....	462
serCwrFlush .....	471	serFopen .....	463
serCwrFree .....	471	serFparity .....	464
serCwrite .....	472	serFpeek .....	465
serCwrUsed .....	473	serFputc .....	466
serDclose .....	456	serFputs .....	467
serDdatabits .....	456	serFrdFlush .....	468
serDdmaOff .....	457	serFrdFree .....	468
serDdmaOn .....	458	serFrdUsed .....	469
serDflowcontrolOff .....	459	serFread .....	470
serDflowcontrolOn .....	460	serFwrFlush .....	471
serDgetc .....	461	serFwrFree .....	471
serDgetError .....	462	serFwrite .....	472
serDopen .....	463	serFwrUsed .....	473
serDparity .....	464	servo_alloc_table .....	438
serDpeek .....	465	servo_closedloop .....	438
serDputc .....	466	servo_disable_0 .....	439
serDputs .....	467	servo_disable_1 .....	440
serDrdFlush .....	468	servo_enable_0 .....	441
serDrdFree .....	468	servo_enable_1 .....	442
serDrdUsed .....	469	servo_gear .....	443
serDread .....	470	servo_graph .....	445
serDwrFlush .....	471	servo_init .....	446
serDwrFree .....	471	servo_millirpm2vcmd .....	446
serDwrite .....	472	servo_move_to .....	447
serDwrUsed .....	473	servo_openloop .....	448
serEclose .....	456	servo_qd_zero_0 .....	449

servo_qd_zero_1 .....	449
servo_read_table .....	450
servo_set_coeffs .....	451
servo_set_pos .....	452
servo_set_vel .....	453
servo_stats_reset .....	453
servo_torque .....	454
serXdatabits .....	456
serXdmaOff .....	457
serXdmaOn .....	458
serXflowcontrolOff .....	459
serXflowcontrolOn .....	460
serXgetc .....	461
serXgetError .....	462
serXparity .....	464
serXpeek .....	465
serXputc .....	466
serXputs .....	467
serXrdFlush .....	468
serXrdFree .....	468
serXrdUsed .....	469
serXread .....	470
serXwrFlush .....	471
serXwrFree .....	471
serXwrite .....	472
serXwrUsed .....	473
SET .....	474
set .....	474
set_cpu_power_mode .....	477
set32kHzDivider .....	475
setClockModulation .....	476
setjmp .....	479
SetSerialTATxRValues .....	480
SetVectExtern2000 .....	481
SetVectExtern3000 .....	482
SetVectExtern4000 .....	483
SetVectIntern .....	484
sf_getPageCount .....	486
sf_getPageSize .....	486
sf_init .....	487
sf_initDevice .....	488
sf_isWriting .....	489
sf_pageToRAM .....	489
sf_RAMToPage .....	490
sf_readDeviceRAM .....	491
sf_readPage .....	492
sf_readRAM .....	493
sf_writeDeviceRAM .....	494
sf_writePage .....	495
sf_writeRAM .....	496
sfspi_init .....	496
sin .....	497
sinh .....	497
snprintf .....	498

SPIinit .....	499
SPIRead .....	500
SPIWrite .....	501
SPIWrRd .....	502
sprintf .....	503
sqrt .....	504
srand .....	504
strcat .....	505
strchr .....	506
strcmp .....	507
strcmpi .....	508
strcpy .....	509
strncpy .....	510
strlen .....	511
strncat .....	512
strncpy .....	513
strncpyi .....	514
strncpy .....	515
strpbrk .....	516
strchr .....	517
strspn .....	518
strstr .....	519
strtod .....	520
strtok .....	522
strtol .....	523
sysResetChain .....	525

## T

tan .....	526
tanh .....	527
TAT1R_SetValue .....	528
tm_rd .....	529
tm_wr .....	530
tolower .....	531
toupper .....	531

## U

updateTimers .....	532
use32kHzOsc .....	532
useClockDivider .....	533
useClockDivider3000 .....	534
useMainOsc .....	535
utoa .....	535

## V

VdGetFreeWd .....	537
VdInit .....	538
VdReleaseWd .....	539
vram2root .....	536

## W

write_rtc .....	542
WriteFlash2 .....	540

WriteFlash2Array .....	541
writeUserBlock .....	543
writeUserBlockArray .....	545
WrPortE .....	547
WrPortI .....	548

## **X**

xalloc .....	549
xalloc_stats .....	551
xavail .....	552
xCalculateECC256 .....	554
xChkCorrectECC256 .....	555
xgetfloat .....	556
xgetint .....	556
xgetlong .....	557
xmem2root .....	558
xmem2xmem .....	559
xmemchr .....	560
xmemcmp .....	561
xrelease .....	562
xsetfloat .....	563
xsetint .....	563
xsetlong .....	564
xstrlen .....	564



## Group Listing of Dynamic C Functions

New releases of Dynamic C often contain new API functions. You can check if your version of Dynamic C contains a particular function by checking the Function Lookup feature in the Help menu. If you see functions described in this manual that you want but do not have, please consider updating your version of Dynamic C. To update Dynamic C, go to: [www.rabbit.com/products/dc/](http://www.rabbit.com/products/dc/) or call 1.530.757.8400.

<b>A</b>	ispunct .....	227
	isspace .....	226
	isupper .....	228
	isxdigit .....	228
<b>Arithmetic</b>		
abs .....		32
getcrc .....		200
lsqrt .....		250
<b>B</b>		
<b>Bit Manipulation</b>		
BIT .....		46
bit .....		45
RES .....		419
res .....		419
SET .....		474
set .....		474
<b>Bus Operation (Rabbit 3000, 4000)</b>		
disableIObus .....		75
enableIObus .....		98
<b>C</b>		
<b>Character</b>		
isalnum .....		221
isalpha .....		222
isctrl .....		222
isdigit .....		224
isgraph .....		225
islower .....		225
isprint .....		226
	<b>D</b>	
	<b>Data Encryption</b>	
	AESdecrypt .....	34
	AESdecryptStream .....	34
	AESencrypt .....	35
	AESencryptStream .....	36
	AESexpandKey .....	37
	AESinitStream .....	38
	<b>Direct Memory Access (Rabbit 4000, 5000)</b>	
	DMAalloc .....	76
	DMAcompleted .....	77
	DMAhandle2chan .....	78
	DMAioe2mem .....	79
	DMAioi2mem .....	81
	DMAloadBufDesc .....	82
	DMAmatchSetup .....	83
	DMAmem2ioe .....	84
	DMAmem2ioi .....	85
	DMAmem2mem .....	86
	DMApoll .....	87
	DMAprintBufDesc .....	88
	DMAprintRegs .....	89

DMAsetBufDesc .....	90
DMAsetDirect .....	91
DMAsetParameters .....	92
DMAstartAuto .....	93
DMAstartDirect .....	94
DMAstop .....	95
DMAstopDirect .....	96
DMAtimerSetup .....	96
DMAunalloc .....	97
serAdmaOff .....	457
serAdmaOn .....	458
serBdmaOff .....	457
serBdmaOn .....	458
serCdmaOff .....	457
serCdmaOn .....	458
serDdmaOff .....	457
serDdmaOn .....	458
serEdmaOff .....	457
serEdmaOn .....	458
serFdmaOff .....	457
serFdmaOn .....	458
serXdmaOff .....	457
serXdmaOn .....	458

## Dynamic Memory Allocation

palloc .....	346
palloc_fast .....	347
pavail .....	348
pavail_fast .....	349
pcalloc .....	350
pfirst .....	351
pfirst_fast .....	352
pfree .....	353
pfree_fast .....	354
phwm .....	355
plast .....	363
plast_fast .....	364
pmovebetween .....	365
pmovebetween_fast .....	367

pncl .....	368
pnxnext .....	369
pnxnext_fast .....	370
pool_append .....	372
pool_init .....	373
pool_link .....	374
pool_xappend .....	375
pool_xinit .....	376
pprev .....	379
pprev_fast .....	380
pputlast .....	381
pputlast_fast .....	382
preorder .....	383
pxalloc .....	394
pxalloc_fast .....	395
pxcalloc .....	396
pxfirst .....	397
pxfirst_fast .....	398
pxfree .....	399
pxfree_fast .....	400
pxlast .....	401
pxlast_fast .....	402
pxnext .....	403
pxnext_fast .....	404
pxprev .....	405
pxprev_fast .....	406

## E

### ECC

CalculateECC256 .....	51
ChkCorrectECC256 .....	52
xCalculateECC256 .....	554
xChkCorrectECC256 .....	555

### Error Handling

errlogFormatEntry .....	100
errlogFormatRegDump .....	101
errlogFormatStackDump .....	101
errlogGetHeaderInfo .....	99



errlogGetMessage .....	102
errlogGetNthEntry .....	100
errlogReadHeader .....	102
error_message .....	103
exception .....	104
ResetErrorLog .....	420

## Extended Memory

_xalloc .....	550
_xavail .....	553
paddr .....	343
paddrDS .....	344
paddrSS .....	345
root2xmem .....	421
xalloc .....	549
xalloc_stats .....	551
xavail .....	552
xgetfloat .....	556
xgetint .....	556
xgetlong .....	557
xmem2root .....	558
xmem2xmem .....	559
xmemchr .....	560
xmemcmp .....	561
xrelease .....	562
xsetfloat .....	563
xsetint .....	563
xsetlong .....	564
xstrlen .....	564

## F

### Fast Fourier Transforms

fftcplx .....	157
fftcplxinv .....	158
fftreal .....	159
fftrealinv .....	160
hanncplx .....	208
hannreal .....	209
powerspectrum .....	378

## File Compression

CloseInputCompressedFile .....	57
CloseOutputCompressedFile .....	57
CompressFile .....	66
DecompressFile .....	70
OpenInputCompressedFile .....	280
OpenOutputCompressedFile .....	281
ReadCompressedFile .....	415

## File System, FAT

fat_AutoMount .....	107
fat_Close .....	110
fat_CreateDir .....	111
fat_CreateFile .....	112
fat_CreateTime .....	113
fat_Delete .....	114
fat_EnumDevice .....	115
fat_EnumPartition .....	116
fat_FileSize .....	117
fat_FormatDevice .....	118
fat_FormatPartition .....	119
fat_Free .....	120
fat_GetAttr .....	121
fat_GetName .....	122
fat_Init .....	123
fat_InitUCOSMutex .....	124
fat_LastAccess .....	125
fat_LastWrite .....	126
fat_MountPartition .....	127
fat_Open .....	128
fat_OpenDir .....	130
fat_PartitionDevice .....	131
fat_Read .....	132
fat_ReadDir .....	133
fat_Seek .....	135
fat_SetAttr .....	137
fat_Split .....	138
fat_Status .....	139
fat_SyncFile .....	140

fat_SyncPartition .....	141
fat_Tell .....	142
fat_tick .....	143
fat_Truncate .....	144
fat_UnmountDevice .....	145
fat_UnmountPartition .....	146
fat_Write .....	147
fat_xWrite .....	148

## File System, FS1

fcreate .....	150
fcreate_unused .....	152
fdelete .....	154
fopen_rd .....	170
fopen_wr .....	172
fread .....	174
fs_format .....	177
fs_init .....	179
fs_reserve_blocks .....	181
fsck .....	181
fseek .....	182
ftell .....	193
fwrite .....	197

## File System, FS2

fclose .....	149
fcreate .....	151
fcreate_unused .....	153
fdelete .....	155
fflush .....	156
fopen_rd .....	171
fopen_wr .....	173
fread .....	175
fs_format .....	178
fs_get_flash_lx .....	184
fs_get_lx .....	185
fs_get_lx_size .....	186
fs_get_other_lx .....	187
fs_get_ram_lx .....	188

fs_init .....	180
fs_set_lx .....	189
fs_setup .....	190
fs_sync .....	192
fseek .....	183
fshift .....	195
ftell .....	194
fwrite .....	198
lx_format .....	252

## Flash, NAND

nf_eraseBlock .....	269
nf_getPageCount .....	270
nf_getPageSize .....	271
nf_initDevice .....	272
nf_InitDriver .....	274
nf_isBusyRBHW .....	275
nf_isBusyStatus .....	276
nf_readPage .....	277
nf_writePage .....	278
nf_XD_Detect .....	279

## Flash, Parallel

flash_erasechip .....	161
flash_erasector .....	162
flash_gettype .....	163
flash_init .....	164
flash_read .....	165
flash_readsector .....	166
flash_sector2xwindow .....	167
flash_writesector .....	168
WriteFlash2 .....	540
WriteFlash2Array .....	541

## Flash, SD

sdspi_debounce .....	423
sdspi_get_csd .....	424
sdspi_get_scr .....	425
sdspi_get_status_reg .....	426
sdspi_getSectorCount .....	426

sdspi_init_card .....	427	cos .....	69
sdspi_initDevice .....	428	cosh .....	69
sdspi_isWriting .....	429	deg .....	72
sdspi_notbusy .....	429	exp .....	105
sdspi_print_dev .....	430	fabs .....	106
sdspi_process_command .....	431	floor .....	169
sdspi_read_sector .....	432	fmod .....	169
sdspi_reset_card .....	433	frexp .....	176
sdspi_sendingAP .....	434	labs .....	236
sdspi_set_block_length .....	435	ldexp .....	237
sdspi_setLED .....	434	log .....	237
sdspi_write_sector .....	437	log10 .....	248
sdspi_WriteContinue .....	436	modf .....	268
<b>Flash, Serial</b>		poly .....	371
sf_getPageCount .....	486	pow .....	377
sf_getPageSize .....	486	pow10 .....	377
sf_init .....	487	rad .....	411
sf_initDevice .....	488	rand .....	411
sf_isWriting .....	489	randb .....	412
sf_pageToRAM .....	489	randg .....	412
sf_RAMToPage .....	490	sin .....	497
sf_readDeviceRAM .....	491	sinh .....	497
sf_readPage .....	492	sqrt .....	504
sf_readRAM .....	493	srand .....	504
sf_writeDeviceRAM .....	494	tan .....	526
sf_writePage .....	495	tanh .....	527
sf_writeRAM .....	496		
sfspi_init .....	496		
<b>Floating-Point Math</b>		<b>G</b>	
acos .....	32	<b>Global Positioning System</b>	
acot .....	33	gps_get_position .....	206
acsc .....	33	gps_get_utc .....	207
asec .....	39	gps_ground_distance .....	207
asin .....	39		
atan .....	40	<b>H</b>	
atan2 .....	41	<b>HDLC Protocol (Rabbit 3000, 4000, 5000)</b>	
ceil .....	53	HDLCAbortE .....	210
		HDLCAbortF .....	210

HDLCcloseE .....	210
HDLCcloseF .....	210
HDLCdropE .....	211
HDLCdropF .....	211
HDLCerrorE .....	211
HDLCerrorF .....	211
HDLCextClockE .....	212
HDLCextClockF .....	212
HDLCopenE .....	213
HDLCopenF .....	213
HDLCpeekE .....	214
HDLCpeekF .....	214
HDLCreceiveE .....	215
HDLCreceiveF .....	215
HDLCsendE .....	216
HDLCsendF .....	216
HDLCsendingE .....	217
HDLCsendingF .....	217

## I

### I/O

BitRdPortE .....	47
BitRdPortI .....	48
BitWrPortE .....	49
BitWrPortI .....	50
RdPortE .....	413
RdPortI .....	414
WrPortE .....	547
WrPortI .....	548

### I2C Protocol

i2c_check_ack .....	230
i2c_init .....	231
i2c_read_char .....	231
i2c_send_ack .....	232
i2c_send_nak .....	232
i2c_start_tx .....	233
i2c_startw_tx .....	234
i2c_stop_tx .....	235

i2c_write_char .....	235
----------------------	-----

### Interrupts

GetVectExtern2000 .....	204
GetVectExtern3000 .....	205
GetVectIntern .....	206
ipres .....	220
ipset .....	221
SetVectExtern2000 .....	481
SetVectExtern3000 .....	482
SetVectExtern4000 .....	483
SetVectIntern .....	484

## L

### Logging Subsystem

log_clean .....	238
log_close .....	239
log_condition .....	240
log_format .....	241
log_map .....	242
log_next .....	243
log_open .....	244
log_prev .....	245
log_put .....	246
log_seek .....	247

## M

### MD5

md5_append .....	259
md5_finish .....	260
md5_init .....	259

### MicroC/OS-II

OOSQDel .....	310
OS_ENTER_CRITICAL .....	282
OS_EXIT_CRITICAL .....	282
OSFlagAccept .....	283
OSFlagCreate .....	285
OSFlagDel .....	286

OSFlagPend .....	287	OSTaskChangePrio .....	323
OSFlagPost .....	289	OSTaskCreate .....	324
OSFlagQuery .....	290	OSTaskCreateExt .....	325
OSInit .....	291	OSTaskCreateHook .....	326
OSMboxAccept .....	291	OSTaskDel .....	327
OSMboxCreate .....	292	OSTaskDelHook .....	328
OSMboxDel .....	293	OSTaskDelReq .....	329
OSMboxPend .....	294	OSTaskIdleHook .....	330
OSMboxPost .....	295	OSTaskQuery .....	330
OSMboxPostOpt .....	296	OSTaskResume .....	331
OSMboxQuery .....	297	OSTaskStatHook .....	331
OSMemCreate .....	298	OSTaskStkChk .....	332
OSMemGet .....	299	OSTaskSuspend .....	333
OSMemPut .....	300	OSTaskSwHook .....	334
OSMemQuery .....	301	OSTCBInitHook .....	334
OSMutexAccept .....	302	OSTimeDly .....	335
OSMutexCreate .....	303	OSTimeDlyHMSM .....	336
OSMutexDel .....	304	OSTimeDlyResume .....	337
OSMutexPend .....	305	OSTimeDlySec .....	338
OSMutexPost .....	306	OSTimeGet .....	339
OSMutexQuery .....	307	OSTimeSet .....	339
OSQAccept .....	308	OSTimeTick .....	340
OSQCreate .....	309	OSTimeTickHook .....	340
OSQFlush .....	311	OSVersion .....	341
OSQPend .....	312		
OSQPost .....	313	<b>Miscellaneous</b>	
OSQPostFront .....	314	hexstrtobyte .....	217
OSQPostOpt .....	315	longjmp .....	248
OSQQuery .....	316	qsort .....	410
OSSchedLock .....	317	runwatch .....	423
OSSchedUnlock .....	317	setjmp .....	479
OSSemAccept .....	318		
OSSemCreate .....	319	<b>Multitasking</b>	
OSSemPend .....	319	CoBegin .....	58
OSSemPost .....	320	CoPause .....	67
OSSemQuery .....	321	CoReset .....	67
OSSetTickPerSec .....	322	CoResume .....	68
OSSStart .....	322	DelayMs .....	73
OSStatInit .....	323	DelaySec .....	74

DelayTicks .....	74
IntervalMs .....	219
IntervalSec .....	219
IntervalTick .....	220
isCoDone .....	223
isCoRunning .....	223
loophead .....	249
loopinit .....	249

## N

### Number-to-String Conversion

ftoa .....	199
htoa .....	218
itoa .....	229
ltoa .....	250
ltoan .....	251
utoa .....	535

## P

### Partitions

mbr_CreatePartition .....	253
mbr_EnumDevice .....	254
mbr_FormatDevice .....	255
mbr_MountPartition .....	256
mbr_UnmountPartition .....	257
mbr_ValidatePartitions .....	258

### Pulse Width Modulation (Rabbit 3000, 4000, 5000)

pwm_init .....	392
pwm_set .....	393

## Q

### Quadrature Decoder (Rabbit 3000, 4000, 5000)

qd_error .....	407
qd_init .....	408
qd_read .....	409
qd_zero .....	409

## R

### Rabbit 3000, 4000

disableIObus .....	75
enableIObus .....	98
servo_alloc_table .....	438
servo_closedloop .....	438
servo_disable_0 .....	439
servo_disable_1 .....	440
servo_enable_0 .....	441
servo_enable_1 .....	442
servo_gear .....	443
servo_graph .....	445
servo_init .....	446
servo_millirpm2vcmd .....	446
servo_move_to .....	447
servo_openloop .....	448
servo_qd_zero_0 .....	449
servo_qd_zero_1 .....	449
servo_read_table .....	450
servo_set_coeffs .....	451
servo_set_pos .....	452
servo_set_vel .....	453
servo_stats_reset .....	453
servo_torque .....	454

### Rabbit 3000, 4000, 5000

cof_pktEreceive .....	58
cof_pktEsend .....	59
cof_pktFreceive .....	58
cof_pktFsend .....	59
cof_serEgetc .....	60
cof_serEgets .....	61
cof_serEputc .....	62
cof_serEputs .....	63
cof_serEread .....	64
cof_serEwrite .....	65
cof_serFgetc .....	60
cof_serFgets .....	61

cof_serFputc .....	62	DMAmem2ioe .....	84
cof_serFputs .....	63	DMAmem2ioi .....	85
cof_serFread .....	64	DMAmem2mem .....	86
cof_serFwrite .....	65	DMApoll .....	87
HDLCabortE .....	210	DMAprintBufDesc .....	88
HDLCabortF .....	210	DMAprintRegs .....	89
HDLCcloseE .....	210	DMAsetBufDesc .....	90
HDLCcloseF .....	210	DMAsetDirect .....	91
HDLCdropE .....	211	DMAsetParameters .....	92
HDLCdropF .....	211	DMAstartAuto .....	93
HDLCerrorE .....	211	DMAstartDirect .....	94
HDLCerrorF .....	211	DMAstop .....	95
HDLCextClockE .....	212	DMAstopDirect .....	96
HDLCextClockF .....	212	DMAtimerSetup .....	96
HDLCopenE .....	213	DMAunalloc .....	97
HDLCopenF .....	213	root2vram .....	420
HDLCpeekE .....	214	serAdmaOff .....	457
HDLCpeekF .....	214	serAdmaOn .....	458
HDLCreceiveE .....	215	serBdmaOff .....	457
HDLCreceiveF .....	215	serBdmaOn .....	458
HDLCsendE .....	216	serCdmaOff .....	457
HDLCsendF .....	216	serCdmaOn .....	458
HDLCsendingE .....	217	serDdmaOff .....	457
HDLCsendingF .....	217	serDdmaOn .....	458
pwm_init .....	392	serEdmaOff .....	457
pwm_set .....	393	serEdmaOn .....	458
qd_error .....	407	serFdmaOff .....	457
qd_init .....	408	serFdmaOn .....	458
qd_read .....	409	serXdmaOff .....	457
qd_zero .....	409	serXdmaOn .....	458
		vram2root .....	536
<b>Rabbit 4000, 5000</b>			
DMAalloc .....	76	<b>Real-Time Clock</b>	
DMAcompleted .....	77	mktime .....	266
DMAhandle2chan .....	78	mktm .....	267
DMAioe2mem .....	79	read_rtc .....	416
DMAioi2mem .....	81	read_rtc_32kHz .....	416
DMAloadBufDesc .....	82	rtc_timezone .....	422
DMAmatchSetup .....	83	set32kHzDivider .....	475

tm_rd .....	529
tm_wr .....	530
updateTimers .....	532
use32kHzOsc .....	532
write_rtc .....	542

## S

### Serial Communication

cof_serAgetc .....	60
cof_serAgets .....	61
cof_serAputc .....	62
cof_serAputs .....	63
cof_serAread .....	64
cof_serAwrite .....	65
cof_serBgetc .....	60
cof_serBgets .....	61
cof_serBputc .....	62
cof_serBputs .....	63
cof_serBread .....	64
cof_serBwrite .....	65
cof_serCgetc .....	60
cof_serCgets .....	61
cof_serCputc .....	62
cof_serCputs .....	63
cof_serCread .....	64
cof_serCwrite .....	65
cof_serDgetc .....	60
cof_serDgets .....	61
cof_serDputc .....	62
cof_serDputs .....	63
cof_serDread .....	64
cof_serDwrite .....	65
cof_serEgetc .....	60
cof_serEgets .....	61
cof_serEputc .....	62
cof_serEputs .....	63
cof_serEread .....	64
cof_serEwrite .....	65

cof_serFgetc .....	60
cof_serFgets .....	61
cof_serFputc .....	62
cof_serFputs .....	63
cof_serFread .....	64
cof_serFwrite .....	65
serAclose .....	456
serAdatabits .....	456
serAdmaOff .....	457
serAdmaOn .....	458
serAflowcontrolOn .....	460
serAgetc .....	461
serAgetError .....	462
serAopen .....	463
serAparity .....	464
serApeek .....	465
serAputc .....	466
serAputs .....	467
serArdFlush .....	468
serArdFree .....	468
serArdUsed .....	469
serAread .....	470
serAwrFlush .....	471
serAwrFree .....	471
serAwrite .....	472
serAwrUsed .....	473
serBclose .....	456
serBdatabits .....	456
serBdmaOff .....	457
serBdmaOn .....	458
serBflowcontrolOn .....	460
serBgetc .....	461
serBgetError .....	462
serBopen .....	463
serBparity .....	464
serBpeek .....	465
serBputc .....	466
serBputs .....	467
serBrdFlush .....	468



serBrdFree .....	468	serDputc .....	466
serBrdUsed .....	469	serDputs .....	467
serBread .....	470	serDrdFlush .....	468
serBwrFlush .....	471	serDrdFree .....	468
serBwrFree .....	471	serDrdUsed .....	469
serBwrite .....	472	serDread .....	470
serBwrUsed .....	473	serDwrFlush .....	471
serCclose .....	456	serDwrFree .....	471
serCdatabits .....	456	serDwrite .....	472
serCdmaOff .....	457	serDwrUsed .....	473
serCdmaOn .....	458	serEclose .....	456
serCflowcontrolOn .....	460	serEdatabits .....	456
serCgetc .....	461	serEdmaOff .....	457
serCgetError .....	462	serEdmaOn .....	458
serCheckParity .....	455	serEflowcontrolOff .....	459
serCopen .....	463	serEflowcontrolOn .....	460
serCparity .....	464	serEgetc .....	461
serCpeek .....	465	serEgetError .....	462
serCputc .....	466	serEopen .....	463
serCputs .....	467	serEparity .....	464
serCrdFlush .....	468	serEpeek .....	465
serCrdFree .....	468	serEputc .....	466
serCrdUsed .....	469	serEputs .....	467
serCread .....	470	serErdFlush .....	468
serCwrFlush .....	471	serErdFree .....	468
serCwrFree .....	471	serErdUsed .....	469
serCwrite .....	472	serEread .....	470
serCwrUsed .....	473	serEwrFlush .....	471
serDclose .....	456	serEwrFree .....	471
serDdatabits .....	456	serEwrite .....	472
serDdmaOff .....	457	serEwrUsed .....	473
serDdmaOn .....	458	serFclose .....	456
serDflowcontrolOff .....	459	serFdatabits .....	456
serDflowcontrolOn .....	460	serFdmaOff .....	457
serDgetc .....	461	serFdmaOn .....	458
serDgetError .....	462	serFflowcontrolOff .....	459
serDopen .....	463	serFflowcontrolOn .....	460
serDparity .....	464	serFgetc .....	461
serDpeek .....	465	serFgetError .....	462

serFopen .....	463
serFparity .....	464
serFpeek .....	465
serFputc .....	466
serFputs .....	467
serFrdFlush .....	468
serFrdFree .....	468
serFrdUsed .....	469
serFread .....	470
serFwrFlush .....	471
serFwrFree .....	471
serFwrite .....	472
serFwrUsed .....	473
serXdatabits .....	456
serXdmaOff .....	457
serXdmaOn .....	458
serXflowcontrolOff .....	459
serXflowcontrolOn .....	460
serXgetc .....	461
serXgetError .....	462
serXparity .....	464
serXpeek .....	465
serXputc .....	466
serXputs .....	467
serXrdFlush .....	468
serXrdFree .....	468
serXrdUsed .....	469
serXread .....	470
serXwrFlush .....	471
serXwrFree .....	471
serXwrite .....	472
serXwrUsed .....	473

## Serial Packet Driver

cof_pktAreceive .....	58
cof_pktAsend .....	59
cof_pktBreceive .....	58
cof_pktBsend .....	59
cof_pktCreceive .....	58

cof_pktCsend .....	59
cof_pktDreceive .....	58
cof_pktDsend .....	59
cof_pktEreceive .....	58
cof_pktEsend .....	59
cof_pktFreceive .....	58
cof_pktFsend .....	59
pktAclose .....	356
pktAgetErrors .....	356
pktAinitBuffers .....	357
pktAopen .....	358
pktAreceive .....	360
pktAsend .....	361
pktAsending .....	362
pktAsetParity .....	362
pktBclose .....	356
pktBgetErrors .....	356
pktBinitBuffers .....	357
pktBopen .....	358
pktBreceive .....	360
pktBsend .....	361
pktBsending .....	362
pktBsetParity .....	362
pktCclose .....	356
pktCgetErrors .....	356
pktCinitBuffers .....	357
pktCopen .....	358
pktCreceive .....	360
pktCsend .....	361
pktCsending .....	362
pktCsetParity .....	362
pktDclose .....	356
pktDgetErrors .....	356
pktDinitBuffers .....	357
pktDopen .....	358
pktDreceive .....	360
pktDsend .....	361
pktDsending .....	362
pktDsetParity .....	362

pktEclose .....	356
pktEgetErrors .....	356
pktEinitBuffers .....	357
pktEopen .....	358
pktEreceive .....	360
pktEsend .....	361
pktEsending .....	362
pktEsetParity .....	362
pktFclose .....	356
pktFgetErrors .....	356
pktFinitBuffers .....	357
pktFopen .....	358
pktFreceive .....	360
pktFsend .....	361
pktFsending .....	362
pktFsetParity .....	362

### Servo Control (Rabbit 3000, 4000)

servo_alloc_table .....	438
servo_closedloop .....	438
servo_disable_0 .....	439
servo_disable_1 .....	440
servo_enable_0 .....	441
servo_enable_1 .....	442
servo_gear .....	443
servo_graph .....	445
servo_init .....	446
servo_millirpm2vcmd .....	446
servo_move_to .....	447
servo_openloop .....	448
servo_qd_zero_0 .....	449
servo_qd_zero_1 .....	449
servo_read_table .....	450
servo_set_coeffs .....	451
servo_set_pos .....	452
servo_set_vel .....	453
servo_stats_reset .....	453
servo_torque .....	454

### SPI

SPIinit .....	499
SPIRead .....	500
SPIWrite .....	501
SPIWrRd .....	502

### Stdio

getchar .....	199
gets .....	201
kbhit .....	236
outchr .....	341
outstr .....	342
printf .....	385
putchar .....	391
puts .....	391
snprintf .....	498
sprintf .....	503

### String Manipulation

memchr .....	261
memcmp .....	262
memcpy .....	263
memmove .....	264
memset .....	265
streat .....	505
strchr .....	506
strcmp .....	507
strcmpi .....	508
strcpy .....	509
strcspn .....	510
strlen .....	511
strncat .....	512
strncmp .....	513
strncmpi .....	514
strncpy .....	515
strpbrk .....	516
strrchr .....	517
strspn .....	518
strstr .....	519

strtok .....	522
tolower .....	531
toupper .....	531

## String-to-Number Conversion

atof .....	42
atoi .....	43
atol .....	44
strtod .....	520
strtol .....	523

## System

_GetSysMacroIndex .....	202
_GetSysMacroValue .....	203
_sysIsSoftReset .....	525
chkHardReset .....	54
chkSoftReset .....	54
chkWDTO .....	55
clockDoublerOff .....	56
clockDoublerOn .....	56
defineErrorHandler .....	71
exit .....	105
forceSoftReset .....	174
getdivider19200 .....	200
GetVectExtern2000 .....	204
GetVectExtern3000 .....	205
GetVectIntern .....	206
ipres .....	220
ipset .....	221
premain .....	382
set_cpu_power_mode .....	477
set32kHzDivider .....	475
setClockModulation .....	476
SetSerialTATxRValues .....	480
sysResetChain .....	525
TAT1R_SetValue .....	528
updateTimers .....	532
use32kHzOsc .....	532
useClockDivider .....	533

useClockDivider3000 .....	534
useMainOsc .....	535

## U

### User Block

readUserBlock .....	417
readUserBlockArray .....	418
writeUserBlock .....	543
writeUserBlockArray .....	545

## V

### VBAT RAM (Rabbit 4000, 5000)

root2vram .....	420
vram2root .....	536

## W

### Watchdogs

Disable_HW_WDT .....	75
Enable_HW_WDT .....	98
hitwd .....	218
VdGetFreeWd .....	537
VdHitWd .....	538
VdInit .....	538
VdReleaseWd .....	539

# 1. Function Descriptions

This chapter includes detailed descriptions for Dynamic C API functions. Not all API functions are included. For example, board-specific functions are described in the board's user manual.

New releases of Dynamic C often contain new API functions. You can check if your version of Dynamic C contains a particular function by checking the Function Lookup feature in the Help menu. If you see functions described in this manual that you want but do not have, please consider updating your version of Dynamic C. To update Dynamic C, go to: [www.rabbit.com/products/dc/](http://www.rabbit.com/products/dc/) or call 1.530.757.8400.

---

---

## abs

---

---

```
int abs ( int x );
```

### DESCRIPTION

Computes the absolute value of an integer argument.

### PARAMETERS

**x**                    Integer argument

### RETURN VALUE

Absolute value of the argument.

### LIBRARY

MATH.LIB

### SEE ALSO

fabs

---

---

## acos

---

---

```
float acos ( float x );
```

### DESCRIPTION

Computes the arccosine of real float value x.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

**x**                    Assumed to be between -1 and 1.

### RETURN VALUE

Arccosine of the argument in radians.

If x is out of bounds, the function returns 0 and signals a domain error.

### LIBRARY

MATH.LIB

### SEE ALSO

cos, cosh, asin, atan

---

---

## acot

---

---

```
float acot( float x );
```

### DESCRIPTION

Computes the arcotangent of real `float` value `x`.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

`x` Assumed to be between `-INF` and `+INF`.

### RETURN VALUE

Arccotangent of the argument in radians.

### LIBRARY

`MATH.LIB`

### SEE ALSO

`tan`, `atan`

---

---

## acsc

---

---

```
float acsc( float x );
```

### DESCRIPTION

Computes the arccosecant of real `float` value `x`.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

`x` Assumed to be between `-INF` and `+INF`.

### RETURN VALUE

The arccosecant of the argument in radians.

### LIBRARY

`MATH.LIB`

### SEE ALSO

`sin`, `asin`

---

---

## AESdecrypt

---

---

```
void AESdecrypt( char * data, char * expandedkey, int nb, int nk );
```

### DESCRIPTION

Decrypts a block of data using an implementation of the Rijndael AES cipher. The encrypted block of data is overwritten by the decrypted block of data.

### PARAMETERS

<b>data</b>	A block of data to be decrypted.
<b>expandedkey</b>	A set of round keys (generated by <code>AESexpandKey()</code> ).
<b>nb</b>	The block size to use. Block is $4 * nb$ bytes long.
<b>nk</b>	The key size to use. Cipher key is $4 * nk$ bytes long.

### LIBRARY

`AES_CRYPT.LIB`

---

---

## AESdecryptStream

---

---

```
void AESdecryptStream( AESstreamState * state, char * data, int count );
```

### DESCRIPTION

Decrypts an array of bytes using the Rabbit implementation of cipher feedback mode. See `Samples\Crypt\AES_STREAMTEST.C` for a sample program and a detailed explanation of the encryption/decryption process.

### PARAMETERS

<b>state</b>	The <code>AESstreamState</code> structure. This memory must be allocated in the program code before calling <code>AESdecryptStream()</code> : <pre>static AESstreamState decrypt_state;</pre>
<b>data</b>	An array of bytes that will be decrypted in place.
<b>count</b>	Size of data array

### LIBRARY

`AES_CRYPT.LIB`



---

---

## AEEncrypt

---

---

```
void AEEncrypt( char * data, char * expandedkey, int nb, int nk );
```

### DESCRIPTION

Encrypts a block of data using an implementation of the Rijndael AES cipher. The block of data is overwritten by the encrypted block of data.

### PARAMETERS

<b>data</b>	A block of data to be encrypted
<b>expandedkey</b>	A set of round keys (generated by <code>AEEexpandKey()</code> )
<b>nb</b>	The block size to use. Block is $4 * nb$ bytes long
<b>nk</b>	The key size to use. Cipher key is $4 * nk$ bytes long

### RETURN VALUE

None.

### LIBRARY

`AES_CRYPT.LIB`

---

---

## AEEncryptStream

---

---

```
void AEEncryptStream( AESstreamState * state, char * data, int count
    );
```

### DESCRIPTION

Encrypts an array of bytes using the Rabbit implementation of cipher feedback mode. See `Samples\Crypt\AES_STREAMTEST.C` for a sample program and a detailed explanation of the encryption/decryption process.

### PARAMETERS

<b>state</b>	The <code>AESstreamState</code> structure. This memory must be allocated in the program code before calling <code>AEEncryptStream()</code> : <pre>static AESstreamState encrypt_state;</pre>
<b>data</b>	An array of bytes that will be encrypted in place.
<b>count</b>	Size of data array.

### LIBRARY

`AES_CRYPT.LIB`

---

---

## AESExpandKey

---

---

```
void AESExpandKey( char * expanded, char * key, int nb, int nk, int
    rounds );
```

### DESCRIPTION

Prepares a key for use by expanding it into a set of round keys. A key is a “password” to decipher encoded data.

### PARAMETERS

<b>expanded</b>	A buffer for storing the expanded key. The size of the expanded key is $4 * nb * (rounds + 1)$ .
<b>key</b>	The cipher key, the size should be $4 * nk$
<b>nb</b>	The block size will be $4 * nb$ bytes long.
<b>nk</b>	The key size will be $4 * nk$ bytes long.
<b>rounds</b>	The number of cipher rounds to use.

### RETURN VALUE

None.

### LIBRARY

AES\_CRYPT.LIB

---

---

## AESinitStream

---

---

```
void AESinitStream( AESstreamState * state, char * key, char *  
    init_vector );
```

### DESCRIPTION

Sets up `AESstreamState` to begin encrypting or decrypting a stream. Each `AESstreamState` structure can only be used for one direction. See `Samples\Crypt\AES_STREAMTEST.C` for a sample program and a detailed explanation of the encryption/decryption process.

### PARAMETERS

<b>state</b>	An <code>AESstreamState</code> structure to be initialized. This memory must be allocated in the program code before calling <code>AESinitStream()</code> .
<b>key</b>	The 16-byte cipher key, using a null pointer, will prevent an existing key from being recalculated.
<b>init_vector</b>	A 16-byte array representing the initial state of the feedback registers. Both ends of the stream must begin with the same initialization vector.

### RETURN VALUE

None.

### LIBRARY

`AES_CRYPT.LIB`

---

---

## asec

---

---

```
float asec( float x );
```

### DESCRIPTION

Computes the arcsecant of real `float` value `x`.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

`x` Assumed to be between `-INF` and `+INF`.

### RETURN VALUE

The arcsecant of the argument in radians.

### LIBRARY

`MATH.LIB`

### SEE ALSO

`cos`, `acos`

---

---

## asin

---

---

```
float asin( float x );
```

### DESCRIPTION

Computes the arcsine of real `float` value `x`.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

`x` Assumed to be between `-1` and `+1`.

### RETURN VALUE

The arcsine of the argument in radians.

### LIBRARY

`MATH.LIB`

### SEE ALSO

`sin`, `acsc`

---

---

## atan

---

---

```
float atan( float x );
```

### DESCRIPTION

Computes the arctangent of real `float` value `x`.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

`x` Assumed to be between `-INF` and `+INF`.

### RETURN VALUE

The arctangent of the argument in radians.

### LIBRARY

`MATH.LIB`

### SEE ALSO

`tan`, `acot`

---

---

## atan2

---

---

```
float atan2( float y, float x );
```

### DESCRIPTION

Computes the arctangent of real `float` value  $y/x$  to find the angle in radians between the x-axis and the ray through (0,0) and (x,y).

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

**y**                      The point corresponding to the y-axis  
**x**                      The point corresponding to the x-axis

### RETURN VALUE

If both `y` and `x` are zero, the function returns 0 and signals a domain error. Otherwise the arctangent of  $y/x$  is returned as follows:

Returned Value (in Radians)	Parameter Values
<i>angle</i>	$x \neq 0, y \neq 0$
PI/2	$x = 0, y > 0$
-PI/2	$x = 0, y < 0$
0	$x > 0, y = 0$
PI	$x < 0, y = 0$

### LIBRARY

MATH.LIB

### SEE ALSO

`acos`, `asin`, `atan`, `cos`, `sin`, `tan`

---

---

## atof

---

---

**NEAR SYNTAX:** `float _n_atof( char * sptr );`  
**FAR SYNTAX:** `float _f_atof( char far * sptr );`

**Note:** By default, `atof()` is defined to `_n_atof()`.

### DESCRIPTION

ANSI string to float conversion (UNIX compatible).

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**sptr**                      String to convert.

### RETURN VALUE

The converted floating value.

If the conversion is invalid, `_xtoxErr` is set to 1. Otherwise `_xtoxErr` is set to 0.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`atoi`, `atol`, `strtod`



---

---

## atoi

---

---

**NEAR SYNTAX:** `int _n_atoi( char * sptr );`

**FAR SYNTAX:** `int _f_atoi( char far * sptr );`

**Note:** By default, `atoi()` is defined to `_n_atoi()`.

### DESCRIPTION

ANSI string to integer conversion (UNIX compatible).

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**sptr**                      String to convert.

### RETURN VALUE

The converted integer value.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`atol`, `atof`, `strtod`

---

---

## atol

---

---

**NEAR SYNTAX:** `long _n_atol( char * sptr );`  
**FAR SYNTAX:** `long _f_atol( char far * sptr );`

By default, `atol()` is defined to `_n_atol()`.

### DESCRIPTION

ANSI string to long conversion (UNIX compatible).

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**sptr**                      String to convert.

### RETURN VALUE

The converted long integer value.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`atoi`, `atof`, `strtod`

---

---

## **bit**

---

---

```
unsigned int bit( void * address, unsigned int bit );
```

### **DESCRIPTION**

Dynamic C may expand this call inline.

Reads specified bit at memory address. `bit` may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
(* (long *)address >> bit) & 1
```

### **PARAMETERS**

<b>address</b>	Address of byte containing bits 7-0
<b>bit</b>	Bit location where 0 represents the least significant bit

### **RETURN VALUE**

1: Specified bit is set.  
0: Bit is clear.

### **LIBRARY**

UTIL.LIB

### **SEE ALSO**

BIT

---

---

## BIT

---

---

```
unsigned int BIT( void * address, unsigned int bit );
```

### DESCRIPTION

Dynamic C may expand this call inline.

Reads specified bit at memory address. `bit` may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
(*(long *)address>>bit) &1
```

### PARAMETERS

<b>address</b>	Address of byte containing bits 7-0
<b>bit</b>	Bit location where 0 represents the least significant bit

### RETURN VALUE

1: bit is set  
0: bit is clear

### LIBRARY

UTIL.LIB

### SEE ALSO

bit

---

---

## BitRdPortE

---

---

```
root int BitRdPortE( unsigned int port, int bitnumber );
```

### DESCRIPTION

Returns 1 or 0 matching the value of the bit read from the specified external I/O port.

### PARAMETERS

<b>port</b>	Address of external parallel port data register.
<b>bitnumber</b>	Bit to read (0-7).

### RETURN VALUE

0 or 1: The value of the bit read.

### LIBRARY

SYSIO.LIB

### SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, RdPortE, WrPortE,  
BitWrPortE

---

---

## BitRdPortI

---

---

```
int BitRdPortI( int port, int bitnumber );
```

### DESCRIPTION

Returns 1 or 0 matching the value of the bit read from the specified internal I/O port.

### PARAMETERS

<b>port</b>	Address of internal parallel port data register.
<b>bitnumber</b>	Bit to read (0–7).

### RETURN VALUE

0 or 1: The value of the bit read.

### LIBRARY

SYSIO.LIB

### SEE ALSO

RdPortI, WrPortI, BitWrPortI, BitRdPortE, RdPortE, WrPortE,  
BitWrPortE

---

---

## BitWrPortE

---

---

```
void BitWrPortE( unsigned int port, char * portshadow, int value, int
    bitcode );
```

### DESCRIPTION

Updates shadow register at `bitcode` with value (0 or 1) and copies shadow to register.

WARNING! A shadow register is required for this function.

### PARAMETERS

<code>port</code>	Address of external parallel port data register.
<code>portshadow</code>	Reference pointer to a variable to shadow the current value of the register.
<code>value</code>	Value of 0 or 1 to be written to the bit position.
<code>bitcode</code>	Bit position 0–7.

### LIBRARY

SYSIO.LIB

### SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, BitRdPortE, RdPortE,  
WrPortE

---

---

## BitWrPortI

---

---

```
void BitWrPortI( int port, char * portshadow, int value, int
    bitcode );
```

### DESCRIPTION

Updates shadow register at position `bitcode` with `value` (0 or 1); copies shadow to register.

WARNING! A shadow register is required for this function.

### PARAMETERS

<code>port</code>	Address of internal parallel port data register.
<code>portshadow</code>	Reference pointer to a variable to shadow the current value of the register.
<code>value</code>	Value of 0 or 1 to be written to the bit position.
<code>bitcode</code>	Bit position 0–7.

### LIBRARY

`SYSIO.LIB`

### SEE ALSO

`RdPortI`, `BitRdPortI`, `WrPortI`, `BitRdPortE`, `RdPortE`, `WrPortE`,  
`BitWrPortE`



---

---

## CalculateECC256

---

---

```
long CalculateECC256( void * data );
```

### DESCRIPTION

Calculates a 3 byte Error Correcting Checksum (ECC, 1 bit correction and 2 bit detection capability) value for a 256 byte (2048 bit) data buffer located in root memory.

### PARAMETERS

**data**                      Pointer to the 256 byte data buffer

### RETURN VALUE

The calculated ECC in the 3 LSBs of the long (i.e., BCDE) result. Note that the MSB (i.e., B) of the long result is always zero.

### LIBRARY

ECC.LIB (This function was introduced in Dynamic C 9.01)

---

---

## ChkCorrectECC256

---

---

```
void ChkCorrectECC256( void * data, void * old_ecc, void * new_ecc);
```

### DESCRIPTION

Checks the old versus new ECC values for a 256 byte (2048 bit) data buffer, and if necessary and possible (1 bit correction, 2 bit detection), corrects the data in the specified root memory buffer.

### PARAMETERS

<b>data</b>	Pointer to the 256 byte data buffer
<b>old_ecc</b>	Pointer to the old (original) 3 byte ECC's buffer
<b>new_ecc</b>	Pointer to the new (current) 3 byte ECC's buffer

### RETURN VALUE

0: Data and ECC are good (no correction is necessary)  
1: Data is corrected and ECC is good  
2: Data is good and ECC is corrected  
3: Data and/or ECC are bad and uncorrectable

### LIBRARY

ECC.LIB (This function was introduced in Dynamic C 9.01)

---

---

## ceil

---

---

```
float ceil( float x );
```

### DESCRIPTION

Computes the smallest integer greater than or equal to the given number.

### PARAMETERS

**x**                      Number to round up.

### RETURN VALUE

The rounded up number.

### LIBRARY

MATH.LIB

### SEE ALSO

floor, fmod

---

---

## chkHardReset

---

---

```
int chkHardReset( void );
```

### DESCRIPTION

This function determines whether this restart of the board is due to a hardware reset. Asserting the RESET line or recycling power are both considered hardware resets. A watchdog timeout is not a hardware reset.

### RETURN VALUE

1: The processor was restarted due to a hardware reset.  
0: If it was not.

### LIBRARY

SYS.LIB

### SEE ALSO

chkSoftReset, chkWDTO, \_sysIsSoftReset

---

---

## chkSoftReset

---

---

```
int chkSoftReset( void );
```

### DESCRIPTION

This function determines whether this restart of the board is due to a software reset from Dynamic C or a call to `forceSoftReset()`.

### RETURN VALUE

1: The board was restarted due to a soft reset.  
0: If it was not.

### LIBRARY

SYS.LIB

### SEE ALSO

chkHardReset, chkWDTO, \_sysIsSoftReset

---

---

## chkWDTO

---

---

```
int chkWDTO( void );
```

### DESCRIPTION

This function determines whether this restart of the board is due to a watchdog timeout.

**Note:** A watchdog timeout cannot be detected on a BL2000 or SmartStar.

### RETURN VALUE

- 1: If the board was restarted due to a watchdog timeout.
- 0: If it was not.

### LIBRARY

SYS.LIB

### SEE ALSO

chkHardReset, chkSoftReset, \_sysIsSoftReset

---

---

## clockDoublerOn

---

---

```
void clockDoublerOn( void );
```

### DESCRIPTION

Enables the Rabbit clock doubler. If the doubler is already enabled, there will be no effect. Also attempts to adjust the communication rate between Dynamic C and the board to compensate for the frequency change. User serial port rates need to be adjusted accordingly. Also note that single-stepping through this routine will cause Dynamic C to lose communication with the target.

### LIBRARY

SYS.LIB

### SEE ALSO

clockDoublerOff

---

---

## clockDoublerOff

---

---

```
void clockDoublerOff( void );
```

### DESCRIPTION

Disables the Rabbit clock doubler. If the doubler is already disabled, there will be no effect. Also attempts to adjust the communication rate between Dynamic C and the board to compensate for the frequency change. User serial port rates need to be adjusted accordingly. Also note that single-stepping through this routine will cause Dynamic C to lose communication with the target.

### LIBRARY

SYS.LIB

### SEE ALSO

clockDoublerOn

---

---

## CloseInputCompressedFile

---

---

```
void CloseInputCompressedFile( ZFILE * ifp );
```

### DESCRIPTION

Close an input compression file opened by `OpenInputCompressionFile()`. This file may be a compressed file that is being decompressed, or an uncompressed file that is being compressed. In either case, this function should be called for each open import ZFILE once it is done being used to free up the associated input buffer.

### PARAMETERS

`ifp` File descriptor of an input compression ZFILE.

### RETURN VALUE

None

### LIBRARY

LZSS.LIB

---

---

## CloseOutputCompressedFile

---

---

```
void CloseOutputCompressedFile( ZFILE * ifp );
```

### DESCRIPTION

Close an output compression file. This file is an FS2 ZFILE which was previously opened with `OpenOutputCompressionFile()`. This function should always be called when done writing to a compression output ZFILE to free up the associated output buffer.

### PARAMETERS

`ifp` File descriptor of an output compression ZFILE.

### RETURN VALUE

None

### LIBRARY

lzss.lib

---

---

## CoBegin

---

---

```
void CoBegin( CoData * p );
```

### DESCRIPTION

Initialize a costatement structure so the costatement will be executed next time it is encountered.

### PARAMETERS

**p**                      Address of costatement

### LIBRARY

COSTATE.LIB

---

---

## cof\_pktXreceive

---

---

```
int cof_pktXreceive( void * buffer, int buffer_size ); /* X is A-F */
```

### DESCRIPTION

Receives an incoming packet. This function returns after a complete packet has been read into the buffer.

The functions `cof_pktEreceive()` and `cof_pktFreceive()` are available when using the Rabbit 3000 or Rabbit 4000.

### PARAMETERS

**buffer**                A buffer for the packet to be written into.

**buffer\_size**        Length of the buffer.

### RETURN VALUE

>0: The number of bytes in the received packet on success.

0: No new packets have been received.

-1: The packet is too large for the given buffer.

-2: A needed `test_packet` function is not defined.

### LIBRARY

PACKET.LIB



---

---

## cof\_pktXsend

---

---

```
void cof_pktXsend(void *send_buffer int buffer_length, char delay );  
/* X is A-F */
```

### DESCRIPTION

Initiates the sending of a packet of data. The function will exit when the packet is finished transmitting.

The functions `cof_pktEsend()` and `cof_pktFsend()` are available when using the Rabbit 3000 or Rabbit 4000.

### PARAMETERS

<b>send_buffer</b>	The data to be sent.
<b>buffer_length</b>	Length of the data buffer to transmit.
<b>delay</b>	The number of byte times (0-255) to delay before sending data. This is used to implement protocol-specific delays between packets.

### LIBRARY

PACKET.LIB

---

---

## cof\_serXgetc

---

---

```
int cof_serXgetc( void ); /* where X is A-F */
```

### DESCRIPTION

This single-user cofunction yields to other tasks until a character is read from port X. This function only returns when a character is successfully written. It is non-reentrant.

The functions `cof_serEgetc()` and `cof_serFgetc()` may be used with the Rabbit 3000 or Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `cof_serXgetc(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

An integer with the character read into the low byte.

### LIBRARY

`RS232.LIB`

### EXAMPLE

```
// echoes characters
main() {
    int c;
    serXopen(19200);
    loopinit();
    while (1) {
        loophead();
        wfd c = cof_serAgetc();
        wfd cof_serAputc(c);
    }
    serAclose();
}
```

---

---

## cof\_serXgets

---

---

```
int cof_serXgets( char * s, int max, unsigned long tmout );
/* where X is A-F */
```

### DESCRIPTION

This single-user cofunction reads characters from port X until a null terminator, linefeed, or carriage return character is read, max characters are read, or until tmout milliseconds transpires between characters read. A timeout will never occur if no characters have been received. This function is non-reentrant. It yields to other tasks for as long as the input buffer is locked or whenever the buffer becomes empty as characters are read. s will always be null terminated upon return. The functions cof\_serEgets () and cof\_serFgets () may be used with the Rabbit 3000 or Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: cof\_serXgets(int port, ...), where “port” is one of the macros SER\_PORT\_A through SER\_PORT\_F.

### PARAMETERS

<b>s</b>	Character array into which a null terminated string is read.
<b>max</b>	The maximum number of characters to read into s.
<b>tmout</b>	Millisecond wait period between characters before timing out.

### RETURN VALUE

1 if CR or max bytes read into s.  
0 if function times out before reading CR or max bytes.

### LIBRARY

RS232.LIB

### EXAMPLE

```
main() { // echoes null terminated character strings
    int getOk;
    char s[16];
    serAopen(19200);
    loopinit();
    while (1) {
        loophead();
        costate {
            wfd getOk = cof_serAgets (s, 15, 20);
            if (getOk)
                wfd cof_serAputs(s);
            else { // timed out: s null terminated, but incomplete
            }
        }
    }
    serAclose();
}
```

---

---

## cof\_serXputc

---

---

```
void cof_serXputc ( int c ); /* where X is A-F */
```

### DESCRIPTION

This single-user cofunction writes a character to serial port X, yielding to other tasks when the input buffer is locked. This function is non-reentrant.

The functions `cof_serEputc()` and `cof_serFputc()` may be used with the Rabbit 3000 or Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `cof_serXputc(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

**c**                      Character to write.

### LIBRARY

RS232.LIB

### EXAMPLE

```
// echoes characters
main() {
    int c;
    serAopen(19200);
    loopinit();
    while (1) {
        loophead();
        wfd c = cof_serAgetc();
        wfd cof_serAputc(c);
    }
    serAclose();
}
```

---

---

## cof\_serXputs

---

---

```
void cof_serXputs( char * str ); /* where X is A-F */
```

### DESCRIPTION

This single-user cofunction writes a null terminated string to port X. It yields to other tasks for as long as the input buffer may be locked or whenever the buffer may become full as characters are written. This function is non-reentrant.

The functions `cof_serEputs()` and `cof_serFputs()` may be used with the Rabbit 3000 or Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `cof_serXputs(port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

**str**                      Null terminated character string to write.

### LIBRARY

RS232.LIB

### EXAMPLE

```
// writes a null terminated character string, repeatedly
main() {
    const char s[] = "Hello Rabbit";
    serAopen(19200);
    loopinit();
    while (1) {
        loophead();
        costate {
            wfd cof_serAputs(s);
        }
    }
    serAclose();
}
```

---

---

## cof\_serXread

---

---

```
int cof_serXread( void * data, int length, unsigned long tmout );
/* X is A-F */
```

### DESCRIPTION

This single-user cofunction reads `length` characters from port `X` (where `X` is A, B, C, D, E or F) or until `tmout` milliseconds transpires between characters read. It yields to other tasks for as long as the input buffer is locked or whenever the buffer becomes empty as characters are read. A timeout will never occur if no characters have been read. This function is non-reentrant.

The functions `cof_serEread()` and `cof_serFread()` may be used with the Rabbit 3000 or Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `cof_serXread(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

<b>data</b>	Data structure into which characters are read.
<b>length</b>	The number of characters to read into <code>data</code> .
<b>tmout</b>	Millisecond wait period to allow between characters before timing out.

### RETURN VALUE

Number of characters read into `data`.

### LIBRARY

`RS232.LIB`

### EXAMPLE

```
// echoes a block of characters
main() {
    int n;
    char s[16];
    serAopen(19200);
    loopinit();
    while (1) {
        loophead();
        costate {
            wfd n = cof_serAread(s, 15, 20);
            wfd cof_serAwrite(s, n);
        }
    }
    serAclose();
}
```

---

---

## cof\_serXwrite

---

---

```
void cof_serXwrite( void * data, int length ); /* where X is A-F */
```

### DESCRIPTION

This single-user cofunction writes `length` bytes to port X. It yields to other tasks for as long as the input buffer is locked or whenever the buffer becomes full as characters are written. This function is non-reentrant.

The functions `cof_serEwrite()` and `cof_serFwrite()` may be used with the Rabbit 3000 or Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `cof_serXwrite(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

<b>data</b>	Data structure to write.
<b>length</b>	Number of bytes in <code>data</code> to write.

### LIBRARY

RS232.LIB

### EXAMPLE

```
// writes a block of characters, repeatedly
main() {
    const char s[] = "Hello Rabbit";
    serAopen(19200);
    loopinit();
    while (1) {
        loophead();
        costate {
            wfd cof_serAwrite(s, strlen(s));
        }
    }
    serAclose();
}
```

---

---

## CompressFile

---

---

```
void CompressFile( ZFILE * input, ZFILE * output );
```

### DESCRIPTION

This function compresses the input file (uncompressed ZFILE, opened with `OpenInputCompressFile()`) using the LZ compression algorithm. The result is put into a user-specified output file (an empty ZFILE, opened with `OpenOutputCompressedFile()`).

The macro `OUTPUT_COMPRESSION_BUFFERS` must be defined with a positive non-zero value to use `CompressFile()` or a compile-time error will occur. The default value of `OUTPUT_COMPRESSION_BUFFERS` is zero.

### PARAMETERS

<code>input</code>	Input bit file
<code>output</code>	Output bit file

### RETURN VALUE

None

### LIBRARY

`LZSS.LIB`

### SEE ALSO

`OpenInputCompressedFile`, `OpenOutputCompressedFile`



---

---

## CoPause

---

---

```
void CoPause( CoData * p );
```

### DESCRIPTION

Pause execution of a costatement so that it will not run the next time it is encountered unless and until CoResume (p) or CoBegin (p) are called.

### PARAMETERS

**p**                      Address of costatement

### LIBRARY

COSTATE.LIB

---

---

## CoReset

---

---

```
void CoReset( CoData * p );
```

### DESCRIPTION

Initializes a costatement structure so the costatement will not be executed next time it is encountered.

### PARAMETERS

**p**                      Address of costatement

### LIBRARY

COSTATE.LIB

---

---

## CoResume

---

---

```
void CoResume( CoData * p );
```

### DESCRIPTION

Resume execution of a costatement that has been paused.

### PARAMETERS

**p**                      Address of costatement

### LIBRARY

COSTATE.LIB

---

---

## cos

---

---

```
float cos( float x );
```

### DESCRIPTION

Computes the cosine of real float value x.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

**x**                      Angle in radians.

### RETURN VALUE

Cosine of the argument.

### LIBRARY

MATH.LIB

### SEE ALSO

`acos`, `cosh`, `sin`, `tan`

---

---

## cosh

---

---

```
float cosh( float x );
```

### DESCRIPTION

Computes the hyperbolic cosine of real float value x. This functions takes a unitless number as a parameter and returns a unitless number.

### PARAMETERS

**x**                      Value to compute.

### RETURN VALUE

Hyperbolic cosine.

If  $|x| > 89.8$  (approx.), the function returns INF and signals a range error.

### LIBRARY

MATH.LIB

### SEE ALSO

`cos`, `acos`, `sin`, `sinh`, `tan`, `tanh`

---

---

## DecompressFile

---

---

```
void DecompressFile( ZFILE * input, ZFILE * output );
```

### DESCRIPTION

This is the expansion routine for the LZSS algorithm. It performs the opposite operation of `CompressFile()`. The input file (a compressed `ZFILE`, opened with `OpenInputCompressedFile()`) is decompressed to the output file (an empty FS2 `ZFILE`, opened with `OpenOutputCompressedFile()`).

### PARAMETERS

<code>input</code>	Input bit file
<code>output</code>	Output bit file

### RETURN VALUE

None

### LIBRARY

LZSS.LIB

---

---

## defineErrorHandler

---

---

```
void defineErrorHandler( void * errfcn );
```

### DESCRIPTION

Sets the BIOS function pointer for runtime errors to the function pointed to by `errfcn`. This user-defined function must be in root memory. Specify `root` at the start of the function definition to ensure this. When a runtime error occurs, the following information is passed to the error handler on the stack:

Stack Position	Stack Contents
SP+0	Return address for <code>exceptionRet</code>
SP+2	Error code
SP+4	0x0000 (can be used for additional information)
SP+6	XPC when <code>exception()</code> was called (upper byte)
SP+8	Address where <code>exception()</code> was called

### PARAMETERS

**errfcn**            Pointer to user-defined run-time error handler.

### LIBRARY

`SYS.LIB`

---

---

## deg

---

---

```
float deg( float x );
```

### DESCRIPTION

Changes float radians *x* to degrees

### PARAMETERS

**x**                    Angle in radians.

### RETURN VALUE

Angle in degrees (a float).

### LIBRARY

MATH.LIB

### SEE ALSO

rad

---

---

## DelayMs

---

---

```
int DelayMs( long delayms );
```

### DESCRIPTION

Millisecond time mechanism for the costatement `waitfor` constructs. The initial call to this function starts the timing. The function returns zero and continues to return zero until the number of milliseconds specified has passed.

### PARAMETERS

**delayms**            The number of milliseconds to wait.

### RETURN VALUE

- 1: The specified number of milliseconds have elapsed.
- 0: The specified number of milliseconds have not elapsed.

### LIBRARY

COSTATE.LIB

---

---

## DelaySec

---

---

```
int DelaySec( long delaysec );
```

### DESCRIPTION

Second time mechanism for the costatement `waitfor` constructs. The initial call to this function starts the timing. The function returns zero and continues to return zero until the number of seconds specified has passed.

### PARAMETERS

**delaysec**      The number of seconds to wait.

### RETURN VALUE

- 1: The specified number of seconds have elapsed.
- 0: The specified number of seconds have not elapsed.

### LIBRARY

COSTATE.LIB

---

---

## DelayTicks

---

---

```
int DelayTicks( unsigned ticks );
```

### DESCRIPTION

Tick time mechanism for the costatement `waitfor` constructs. The initial call to this function starts the timing. The function returns zero and continues to return zero until the number of ticks specified has passed.

1 tick = 1/1024 second.

### PARAMETERS

**ticks**      The number of ticks to wait.

### RETURN VALUE

- 1: The specified tick delay has elapsed.
- 0: The specified tick delay has not elapsed.

### LIBRARY

COSTATE.LIB



---

---

## Disable\_HW\_WDT

---

---

```
void Disable_HW_WDT( void );
```

### DESCRIPTION

Disables the hardware watchdog timer on the Rabbit processor. Note that the watchdog will be enabled again just by hitting it. The watchdog is hit by the periodic interrupt, which is on by default. This function is useful for special situations such as low power “sleepy mode.”

### LIBRARY

```
SYS.LIB
```

---

---

## disableIObus

---

---

```
void disableIObus( void );
```

### DESCRIPTION

This function disables external I/O bus and normal data bus operations resume on the Rabbit 3000 or Rabbit 4000.

The external I/O bus must be disabled during normal bus operations with other devices and must be enabled during any external I/O bus operation.

This function is non-reentrant.

Port A and B data shadow register values are NOT saved or restored in this function call.

Parallel port A is set to a byte-wide input and parallel port B data direction register (PBDDR) is set to an unknown state, which must be set by the user.

### LIBRARY

```
ExternIO.LIB (was in R3000.LIB prior to DC 10)
```

### SEE ALSO

```
enableIObus
```

---

---

## DMAalloc

---

---

```
dma_chan_t DMAalloc( char channel_mask, int highest );
```

### DESCRIPTION

This function returns a handle to an available channel. The handle contains the channel number and a validation byte to prevent use of an old handle after deallocation.

### PARAMETERS

**channel\_mask** Mask of all the acceptable channels to choose from.

**highest** Bool indicating whether to search for an available channel from 8 or from 0.

### RETURN VALUE

Returns a handle to a DMA channel if one is available. If none are available it returns DMA\_CHANNEL\_NONE.

### LIBRARY

DMA.LIB

### SEE ALSO

DMAunalloc, DMAhandle2chan

---

---

## DMAcompleted

---

---

```
int DMAcompleted( dma_chan_t handle, unsigned int * len );
```

### DESCRIPTION

This function checks to see if a channel is finished with its DMA operation. If complete, the number of bytes transferred in the last operation is returned in \*len (if len is not NULL), and 1 is returned.

### PARAMETERS

<b>handle</b>	Handle for channel to check
<b>len</b>	Pointer to the value to be filled with the number of bytes last transferred

### RETURN VALUE

1: DMA operation is complete  
0: Allocated channel has never been used or is currently running  
-EINVAL: Invalid handle

### LIBRARY

DMA.LIB

### SEE ALSO

DMAstop

---

---

## DMAhandle2chan

---

---

```
int DMAhandle2chan( dma_chan_t handle );
```

### DESCRIPTION

This function checks the validity of a handle and returns the channel number if it is valid.

### PARAMETER

**handle**            Handle to convert to channel number

### RETURN VALUE

0 - 7: Valid channel number

DMA\_CHANNEL\_NONE: The channel is invalid

### LIBRARY

DMA.LIB

### SEE ALSO

DMAalloc, DMAunalloc

---

---

## DMAioe2mem

---

---

```
int DMAioe2mem( dma_chan_t handle, dma_addr_t dest, unsigned int src,
                unsigned int len, unsigned int flags );
```

### DESCRIPTION

This function performs an immediate DMA operation from external I/O to memory.

### PARAMETERS

<b>handle</b>	Handle for channel to use in transfer
<b>dest</b>	Memory destination address
<b>src</b>	External I/O location source address
<b>len</b>	Length to send (cannot equal zero)
<b>flags</b>	Various flag options. <ul style="list-style-type: none"><li>• <b>DMA_F_REPEAT</b> indicates that the transfer will be a cycle</li><li>• <b>DMA_F_INTERRUPT</b> indicates an interrupt will be triggered at the completion of the transfer. The interrupt vector and function must be set up in the user's code.</li><li>• <b>DMA_F_LAST_SPECIAL</b> (only for Ethernet or HDLC peripherals) Internal Source: Status byte written to initial buffer descriptor before last data. Internal Destination: Last byte written to offset address for frame termination. All Others: no effect.</li><li>• <b>DMA_F_SRC_DEC</b> only for transfers with memory source. Indicates the source address should be decremented.</li><li>• <b>DMA_F_DEST_DEC</b> only for transfers with memory destination. Indicates the destination address should be incremented.</li><li>• <b>DMA_F_STOP_MATCH</b> indicates whether or not to stop the dma transfer when a character is reached. The match byte and mask should have previously been set by calling the <code>DMAmatchSetup()</code> function.</li><li>• <b>DMA_F_TIMER</b> indicates the DMA timer will be used. The divisor should have already been set by calling the <code>DMAtimerSetup()</code> function.</li><li>• <b>DMA_F_TIMER_1BPR</b> indicates that the timed transfers will send one byte per request instead of the entire descriptor</li></ul>

---

---

## DMAioe2mem (cont'd)

---

---

Only one of the following flags (if any) should be set. They indicate that the DMA transfer is gated using the named pin:

- DMA\_F\_PD2, DMA\_F\_PE2, DMA\_F\_PE6, DMA\_F\_PD3,  
DMA\_F\_PE3, DMA\_F\_PE7

The following flags indicate the polarity of the gating signal:

- DMA\_F\_FALLING (default), DMA\_F\_RISING, DMA\_F\_LOW,  
DMA\_F\_HIGH

### RETURN VALUE

- 0: Success
- EINVAL: Invalid handle
- EBUSY: Resources are busy

### LIBRARY

DMA.LIB

### SEE ALSO

DMAmem2mem, DMAcompleted, DMAstop

---

---

## DMAioi2mem

---

---

```
int DMAioi2mem( dma_chan_t handle, dma_addr_t dest, unsigned int src,
                unsigned int len, unsigned int flags );
```

### DESCRIPTION

This function performs an immediate DMA operation from internal I/O to memory.

### PARAMETERS

<b>handle</b>	Handle for channel to use in transfer
<b>dest</b>	Memory destination address
<b>src</b>	Internal I/O location source address
<b>len</b>	Length to send (cannot equal zero)
<b>flags</b>	Various flag options. See <a href="#">DMAioe2mem()</a> for a full list of flags and their descriptions.

### RETURN VALUE

0: Success  
-EINVAL: Invalid handle  
-EBUSY: Resources are busy

### LIBRARY

DMA.LIB

### SEE ALSO

DMAmem2mem, DMAcompleted, DMAstop

---

---

## DMAloadBufDesc

---

---

```
void DMAloadBufDesc( int dmaChannel, dma_addr_t * bufPtr );
```

### DESCRIPTION

This function loads the appropriate DMA Initial Address Registers for the requested DMA channel with the address provided.

### PARAMETERS

<b>dmaChannel</b>	DMA channel number to load
<b>bufPtr</b>	Pointer to variable containing physical address of DMA buffer

### LIBRARY

DMA.LIB

### SEE ALSO

DMAsetBufDesc, DMAsetDirect



---

---

## DMAMatchSetup

---

---

```
int DMAMatchSetup( dma_chan_t handle, int mask, int byte );
```

### DESCRIPTION

This function sets up the mask and match registers for the DMA. These registers are only used when the `DMA_F_STOP_MATCH` flag is passed to the transfer function.

### PARAMETERS

<b>handle</b>	Handle for the DMA channel.
<b>mask</b>	Mask for termination byte (parameter 3). A value of all zeros disables the termination byte match feature. A value of all ones uses the full termination byte for comparison.
<b>byte</b>	Byte that, if matched, will terminate the buffer.

### LIBRARY

`DMA.LIB`

### SEE ALSO

`DMAMem2mem`, `DMAtimerSetup`

---

---

## DMAmem2ioe

---

---

```
int DMAmem2ioe( dma_chan_t handle, unsigned int dest, dma_addr_t src,
               unsigned int len, unsigned int flags );
```

### DESCRIPTION

This function performs an immediate DMA operation from memory to external I/O.

### PARAMETERS

<b>handle</b>	Handle for channel to use in transfer
<b>dest</b>	External I/O destination address
<b>src</b>	Memory location source
<b>len</b>	Length to send (cannot equal zero)
<b>flags</b>	Various flag options. See <a href="#">DMAioe2mem()</a> for a full list of flags and their descriptions.

### RETURN VALUE

0: Success  
-EINVAL: Invalid handle  
-EBUSY: Resources are busy

### LIBRARY

DMA.LIB

### SEE ALSO

DMAmem2mem, DMAcompleted, DMAstop

---

---

## DMAmem2ioi

---

---

```
int DMAmem2ioi( dma_chan_t handle, unsigned int dest, dma_addr_t src,
                unsigned int len, unsigned int flags );
```

### DESCRIPTION

This function performs an immediate DMA operation from memory to internal I/O.

### PARAMETERS

<b>handle</b>	Handle for channel to use in transfer
<b>dest</b>	Internal I/O destination address
<b>src</b>	Memory location source
<b>len</b>	Length to send (cannot equal zero)
<b>flags</b>	Various flag options. See <a href="#">DMAioe2mem()</a> for a full list of flags and their descriptions.

### RETURN VALUE

0: Success  
-EINVAL: Invalid handle  
-EBUSY: Resources are busy

### LIBRARY

DMA.LIB

### SEE ALSO

DMAmem2mem, DMAcompleted, DMAstop

---

---

## DMAmem2mem

---

---

```
int DMAmem2mem( dma_chan_t handle, dma_addr_t dest, dma_addr_t src,
                unsigned int len, unsigned int flags );
```

### DESCRIPTION

This function performs an immediate DMA operation from memory to memory.

### PARAMETERS

<b>handle</b>	Handle for channel to use in transfer
<b>dest</b>	Memory destination address
<b>src</b>	Memory location source address
<b>len</b>	Length to send (cannot equal zero)
<b>flags</b>	Various flag options. See <a href="#">DMAioe2mem()</a> for a full list of flags and their descriptions.

### RETURN VALUE

0: Success  
-EINVAL: Invalid handle  
-EBUSY: Resources are busy

### LIBRARY

DMA.LIB

### SEE ALSO

DMACompleted, DMAstop

---

---

## DMApoll

---

---

```
word DMApoll( int dmaChannel, word * bufCount );
```

### DESCRIPTION

This is a low-level DMA function for determining how much data has been transferred by the specified DMA channel. Since DMA is asynchronous to the CPU, this returns a lower bound on the actually completed transfer.

**IMPORTANT:** Owing to the way the DMA channels are designed, this function will not give a valid result for the first buffer in a linked list or chain, or if there is only one buffer defined (with no link or array sequencing). To get around this limitation, define the first buffer as a dummy transfer of one byte from memory to the same memory, and link this initial dummy buffer to the desired list or array of buffer descriptors. Take the dummy buffer into account when interpreting the `bufCount` value returned. If you service an interrupt from the dummy buffer completion, you will know when it is valid to poll.

This function is mainly intended for endless DMA loops (e.g., receiving into a circular buffer from a serial port) thus the above restriction should not be too onerous in practice.

### PARAMETERS

<b>dmaChannel</b>	DMA channel number to poll (0-7).
<b>bufCount</b>	Pointer to variable in which the completed buffer count will be written. The return value contains the number of bytes remaining (not yet transferred) in this buffer. The buffer count wraps around modulo 256.

### RETURN VALUE

The number of bytes remaining in the buffer indicated by `*bufCount`. This ranges from 0, if completed, up to the total size of the buffer, if not yet started. If the size of any single transfer was 65536 bytes, then the return value is ambiguous as to whether it means “0” or “65536.”

### LIBRARY

DMA.LIB

### SEE ALSO

DMAloadBufDesc, DMAsetDirect

---

---

## DMAprintBufDesc

---

---

```
void DMAprintBufDesc( void * dr, long dp );
```

### DESCRIPTION

This is a debugging function only. It formats and prints the contents of the buffer descriptor at \*dr or \*dp, using bit 6 of the chanControl field to determine whether to assume a short or long format. If dr is not NULL, then the buffer descriptor is in root memory and \*dr is used. Otherwise, dp is assumed to be the physical address of the buffer descriptor in xmem.

### PARAMETERS

<b>dr</b>	Pointer to buffer descriptor in root memory.
<b>dp</b>	Address of buffer descriptor in physical memory.

### LIBRARY

DMA.LIB

### SEE ALSO

DMAprintRegs

---

---

## DMAprintRegs

---

---

```
void DMAprintRegs( int chan, int masters );
```

### DESCRIPTION

This is a debugging function only. This prints the values of the hardware registers for the specified channel. If `masters` is true, then it also prints the values of the master DMA control registers.

Note that the Source and Destination Address registers are write only and read as zero.

### PARAMETERS

<b>chan</b>	Channel number to print
<b>masters</b>	A bool to determine whether or not to print out the master registers shared between all channels

### LIBRARY

DMA.LIB

### SEE ALSO

DMAprintBufDesc

---

---

## DMAsetBufDesc

---

---

```
int DMAsetBufDesc( char chanControl, unsigned int bufLength,  
    dma_addr_t srcAddress, dma_addr_t destAddress, dma_addr_t  
    linkAddress, dma_addr_t bufPtr, int bufSize );
```

### DESCRIPTION

This function loads a DMA buffer descriptor in memory with the values provided. The buffer needs to be described as either 12 or 16 bytes in size.

### PARAMETERS

<b>chanControl</b>	DMA channel control value
<b>bufLength</b>	DMA buffer length
<b>srcAddress</b>	DMA source address
<b>destAddress</b>	DMA destination address
<b>linkAddress</b>	DMA link address (of next buffer descriptor)
<b>bufPtr</b>	Physical address of buffer descriptor to fill
<b>bufSize</b>	Size of buffer descriptor in bytes (12 or 16 only)

### RETURN VALUE

0: Success  
-EINVAL: Error

### LIBRARY

DMA.LIB

### SEE ALSO

DMAloadBufDesc, DMAsetDirect



---

---

## DMAsetDirect

---

---

```
void DMAsetDirect( int channel, char chanControl, unsigned int
    bufLength, dma_addr_t srcAddress, dma_addr_t destAddress,
    dma_addr_t linkAddress );
```

### DESCRIPTION

This function sets up a DMA channel with the values provided.

### PARAMETERS

<b>channel</b>	DMA channel to set
<b>chanControl</b>	DMA channel control value
<b>bufLength</b>	DMA buffer length
<b>srcAddress</b>	DMA source address
<b>destAddress</b>	DMA destination address
<b>linkAddress</b>	DMA link address (of next buffer descriptor)

### LIBRARY

DMA.LIB

### SEE ALSO

DMAloadBufDesc, DMAsetBufDesc

---

---

## DMAsetParameters

---

---

```
int DMAsetParameters( unsigned int transfer_pri, unsigned int
    interrupt_pri, unsigned int inter_dma_pri, unsigned int
    chunkiness, unsigned int min_cpu_pct );
```

### DESCRIPTION

This function sets up DMA parameters. The `chunkiness` parameter determines the amount of CPU time needed to transfer data according to this chart:

chunkiness	1	2	3	4	8	16	32	64
CPU_cycles	11	15	19	23	39	71	135	263

The `min_cpu_pct` parameter determines the minimum time between bursts and is calculated with this formula:

$$\text{cpu free time} = \frac{(\text{CPU\_cycles} \cdot \text{min\_cpu\_pct})}{(100 - \text{min\_cpu\_pct})}$$

This is then rounded up to the nearest value out of 12, 16, 24, 32, 64, 128, 256, or 512.

### PARAMETERS

- transfer\_pri** DMA transfer priority (0, 1, 2 or 3), transfers can occur when the CPU interrupt priority is less than or equal to this value.
- interrupt\_pri** DMA interrupt priority (0, 1, 2, or 3); a value of 0 will disable the DMA interrupts.
- inter\_dma\_pri** Relative prioritization amongst the DMA channels. It is one of the following constants:
- `DMA_IDP_FIXED` - fixed priorities, with higher channel numbers taking precedence;
  - `DMA_IDP_ROTATE_FINE` - priorities are rotated after every byte transferred;
  - `DMA_IDP_ROTATE_COARSE` - priorities rotated after every transfer request, the size of which is determined by the “`chunkiness`” parameter.
- chunkiness** Maximum transfer burst size. Allowed values are 1, 2, 3, 4, 8, 16, 32, or 64. Other numbers will be rounded down to the nearest allowed value.

---

---

## DMAsetParameters (cont'd)

---

---

**min\_cpu\_pct**      A number between 0 and 100 describing the minimum (worst-case) relative amount of time that the CPU will control the bus versus the DMA time. Internally, this function uses this figure to determine the 'minimum clocks between bursts' hardware setting. The figure will be rounded in favor of the CPU, up to the maximum possible hardware setting.

### RETURN VALUE

0: Success  
-EINVAL: for an error

### LIBRARY

DMA.LIB

---

---

## DMAstartAuto

---

---

```
void DMAstartAuto( int channel );
```

### DESCRIPTION

This function is defined to the following:

```
WrPortI(DMALR, NULL, 1 << channel);
```

Start (using auto-load) the corresponding DMA channel, using the buffer descriptor in memory addressed by the Initial Address Register. This command should only be used after the Initial Address has been loaded.

### PARAMETER

**channel**      DMA channel (obtainable through DMAhandle2chan())

### LIBRARY

DMA.LIB

### SEE ALSO

DMAstartDirect, DMAstopDirect

---

---

## DMAstartDirect

---

---

```
void DMAstartDirect( int channel );
```

### DESCRIPTION

This function is defined to the following:

```
WrtPortI(DMCSR, NULL, 1 << channel);
```

Start (or restart) the corresponding DMA channel using the contents of the DMA channel registers. This command should only be used after all the DMA channel registers have been loaded.

### PARAMETER

**channel**            DMA channel (obtainable through DMAhandle2chan())

### LIBRARY

DMA.LIB

### SEE ALSO

DMAstartAuto, DMAstopDirect

---

---

## DMAstop

---

---

```
int DMAstop( dma_chan_t handle );
```

### DESCRIPTION

Stop a DMA operation started with one of the DMAmem2ioe series functions. DMAcompleted() will return TRUE after for an operation stopped with this function, but with less data length than the original request. It is OK to stop an operation that has currently completed; this has no effect. DMAcompleted() may be called to determine the actual amount of data transferred.

### PARAMETER

Handle for channel to stop.

### RETURN VALUE

0: Success  
-EINVAL: Invalid handle

### LIBRARY

DMA.LIB

### SEE ALSO

DMAcompleted, DMAstopDirect

---

---

## DMAstopDirect

---

---

```
void DMAstopDirect( int channel );
```

### DESCRIPTION

This function is defined to the following:

```
WrtPortI(DMHR, NULL, 1 << channel);
```

Halt the corresponding DMA channel. The DMA registers obtain the current state and the DMA can be restarted using the DMCSR.

### PARAMETER

**channel**            DMA channel (obtainable through DMAhandle2chan())

### LIBRARY

DMA.LIB

### SEE ALSO

DMAstartAuto, DMAstartDirect

---

---

## DMAtimerSetup

---

---

```
void DMAtimerSetup( unsigned int divisor );
```

### DESCRIPTION

This function sets up the DMA 16-bit divisor. To use the divisor, the DMA\_F\_TIMER flag must be passed to the transfer function.

### PARAMETER

**divisor**            16-bit divisor for the DMA timer

### LIBRARY

DMA.LIB

### SEE ALSO

DMAmem2mem, DMAmatchSetup

---

---

## DMAunalloc

---

---

```
int DMAunalloc( dma_chan_t handle );
```

### DESCRIPTION

This function deallocates a handle, effectively closing the DMA channel to which it was associated.

### PARAMETER

**handle**            Handle for DMA channel; returned by DMAalloc().

### RETURN VALUE

0: Success  
-EINVAL: Error

### LIBRARY

DMA.LIB

### SEE ALSO

DMAalloc, DMAhandle2chan

---

---

## Enable\_HW\_WDT

---

---

```
void Enable_HW_WDT( void );
```

### DESCRIPTION

Enables the hardware watchdog timer on the Rabbit processor. The watchdog is hit by the periodic interrupt, which is on by default.

### LIBRARY

SYS.LIB

---

---

## enableIObus

---

---

```
void enableIObus( void );
```

### DESCRIPTION

This function enables external I/O bus operation on the Rabbit 3000 or Rabbit 4000. The external I/O bus must be enabled during any external I/O bus operation and disabled during normal bus operations with other devices.

Parallel port A becomes the I/O data bus and parallel port B bits 7:2 becomes the I/O address bus.

This function is non-reentrant.

Port A and B data shadow register values are NOT saved or restored in this function call.

If the macro PORTA\_AUX\_IO has been previously defined, this function should not be called.

### LIBRARY

ExternIO.LIB (was in R3000.LIB prior to DC 10)

### SEE ALSO

disableIObus



---

---

## errlogGetHeaderInfo

---

---

```
root char* errlogGetHeaderInfo( void );
```

### DESCRIPTION

Reads the error log header and formats the output.

When running stand alone (not talking to Dynamic C), this function reads the header directly from the log buffer. When in debug mode, this function reads the header from the copy in flash.

When a Dynamic C cold boot takes place, the header in RAM is zeroed out to initialize it, but first its contents are copied to an address in the BIOS code before the BIOS in RAM is copied to flash. This means that on the second cold boot, the data structure in flash will be zeroed out. The configuration of the log buffer may still be read, and the log buffer entries are not affected.

Because the exception mechanism resets the processor by causing a watchdog time-out, the number of watchdog time-outs reported by this functions is the number of actual WDTOs plus the number of exceptions.

### RETURN VALUE

A null terminated string containing the header information:

```
Status Byte: 0
#Exceptions: 5
Index last exception: 5
#SW Resets: 2
#HW Resets: 2
#WD Timeouts: 5
```

The string will contain “Header checksum invalid” if a checksum error occurs. The meaning of the status byte is as follows:

```
bit 0   - An error has occurred since deployment
bit 1   - The count of SW resets has rolled over.
bit 2   - The count of HW resets has rolled over.
bit 3   - The count of WDTOs has rolled over.
bit 4   - The count of exceptions has rolled over.
bit 5-7 - Not used
```

The index of the last exception is the index from the start of the error log entries. If this index does not equal the total exception count minus one, the error log entries have wrapped around the log buffer.

### LIBRARY

```
ERRORS.LIB
```

---

---

## errlogGetNthEntry

---

---

```
root int errlogGetNthEntry( int N );
```

### DESCRIPTION

Loads `errLogEntry` structure with Nth entry of the error buffer. This must be called before the functions below that format the output.

### PARAMETERS

**N**                      Index of entry to load into `errLogEntry`

### RETURN VALUE

0: Success, entry checksum okay.  
-1: Failure, entry checksum not okay.

### LIBRARY

`ERRORS.LIB`

---

---

## errlogFormatEntry

---

---

```
root char* errlogFormatEntry( void );
```

### DESCRIPTION

Returns a null terminated string containing the basic information contained in `errLogEntry`:

```
Error type=240  
Address = 00:16aa  
Time: 06/11/2001 20:49:29
```

### RETURN VALUE

The null terminated string described above.

### LIBRARY

`ERRORS.LIB`

---

---

## errlogFormatRegDump

---

---

```
root char* errlogFormatRegDump( void );
```

### DESCRIPTION

Returns a null terminated string containing a register dump using the data in `errLogEntry`:

```
AF=0000,AF'=0000  
HL=00f0,HL'=15e3  
BC=16ce,BC'=1600  
DE=0000,DE'=1731  
IX=d3f1,IY =0560  
SP=d3eb,XPC=0000
```

### RETURN VALUE

The null terminated string described above.

### LIBRARY

ERRORS.LIB

---

---

## errlogFormatStackDump

---

---

```
root char * errlogFormatStackDump( void );
```

### DESCRIPTION

Returns a null terminated string containing a stack dump using the data in `errLogEntry`.

```
Stack Dump:  
0024,04f1,  
d378,c146,  
c400,a108,  
2404,0000,
```

### RETURN VALUE

The null terminated string describe above.

### LIBRARY

ERRORS.LIB

---

---

## errlogGetMessage

---

---

```
root char * errlogGetMessage( void );
```

### DESCRIPTION

Returns a null terminated string containing the 8 byte message in `errLogEntry`.

### RETURN VALUE

A null terminated string.

### LIBRARY

`ERRORS.LIB`

---

---

## errlogReadHeader

---

---

```
root int errlogReadHeader( void );
```

### DESCRIPTION

Reads error log header into the structure `errlogInfo`.

### RETURN VALUE

0: Success, entry checksum OK.  
-1: Failure, entry checksum not OK.

### LIBRARY

`ERRORS.LIB`

---

---

## **error\_message**

---

---

```
unsigned long error_message( int message_index );
```

### **DESCRIPTION**

Returns a physical pointer to a descriptive string for an error code listed in `errno.lib`. The sample program `Samples\ErrorHandling\error_message_test.c` illustrates the use of `error_message()`. The error message strings are defined in `errors.lib`.

### **PARAMETER**

**message\_index**     Positive or negative value of error return code.

### **RETURN VALUE**

Physical address of string, or zero if error code is not listed.

### **LIBRARY**

`ERRORS.LIB`

---

---

## exception

---

---

```
int exception( int errCode );
```

### DESCRIPTION

This function is called by Rabbit libraries when a runtime error occurs. It puts information relevant to the runtime error on the stack and calls the default runtime error handler pointed to by the `ERROR_EXIT` macro. To define your own error handler, see the `defineErrorHandler()` function.

When the error handler is called, the following information will be on the stack:

Location on Stack	Description
SP+0	Return address for error handler call
SP+2	Runtime error code
SP+4	(can be used for additional information)
SP+6	XPC when <code>exception()</code> was called (upper byte)
SP+8	Address where <code>exception()</code> was called from

### RETURN VALUE

Runtime error code passed to it.

### LIBRARY

`ERRORS.LIB`

### SEE ALSO

`defineErrorHandler`

---

---

## exit

---

---

```
void exit( int exitcode );
```

### DESCRIPTION

Stops the program and returns `exitcode` to Dynamic C. Dynamic C uses values above 128 for run-time errors. When not debugging, `exit` will run an infinite loop, causing a watchdog timeout if the watchdog is enabled.

### PARAMETERS

**exitcode**      Error code passed by Dynamic C.

### LIBRARY

`SYS.LIB`

---

---

## exp

---

---

```
float exp( float x );
```

### DESCRIPTION

Computes the exponential of real `float` value `x`.

### PARAMETERS

**x**              Value to compute

### RETURN VALUE

Returns the value of  $e^x$ .

If  $x > 89.8$  (approx.), the function returns INF and signals a range error. If  $x < -89.8$  (approx.), the function returns 0 and signals a range error.

### LIBRARY

`MATH.LIB`

### SEE ALSO

`log`, `log10`, `frexp`, `ldexp`, `pow`, `pow10`, `sqrt`

---

---

## **fabs**

---

---

```
float fabs( float x );
```

### **DESCRIPTION**

Computes the float absolute value of `float x`.

### **PARAMETERS**

**x**                    Value to compute.

### **RETURN VALUE**

`x`, if `x >= 0`,  
else `-x`.

### **LIBRARY**

`MATH.LIB`

### **SEE ALSO**

`abs`



---

---

## fat\_AutoMount

---

---

```
int fat_AutoMount( word flags );
```

### DESCRIPTION

Initializes the drivers in the default drivers configuration list in `fat_config.lib` and enumerates the devices in the default devices configuration list, then mounts partitions on enumerated devices according to the device's default configuration flags, unless overridden by the specified run time configuration flags. Despite its lengthy description, this function makes initializing multiple devices using the FAT library as easy as possible. The first driver in the configuration list becomes the primary driver in the system, if one is not already set up.

After this routine successfully returns, the application can start calling directory and file functions for the devices' mounted partitions.

If devices and/or partitions are not already formatted, this function can optionally format them according to the device's configuration or run time override flags.

This function may be called multiple times, but will not attempt to remount device partitions that it has already mounted. Once a device partition has been mounted by this function, unmounts and remounts must be handled by the application.

Even though this function may be called multiple times, it is not meant to be used as a polling or status function. For example, if you are using removable media such as an SD card, you should call `sdspi_debounce()` to determine when the card is fully inserted into the socket.

There are two arrays of data structures that are populated by calling `fat_AutoMount()`. The array named `fat_part_mounted[]` is an array of pointers to `fat_part` structures. A `fat_part` structure holds information about a specific FAT partition. The other array, `_fat_device_table[]`, is composed of pointers to `mbr_dev` structures. An `mbr_dev` structure holds information about a specific device. Partition and device structures are needed in many FAT function calls to specify the device and partition to be used.

An example of using `fat_part_mounted[]` was shown in the sample program `fat_create.c`. FAT applications will need to scan `fat_part_mounted[]` to locate valid FAT partitions. A valid FAT partition must be identified before any file and directory operations can be performed. These pointers to FAT partitions may be used directly by indexing into the array or stored in a local pointer. The `fat_shell.c` sample uses an index into the array, whereas most other sample programs make a copy of the pointer.

An example of using `_fat_device_table[]` is in the sample program `fat_shell.c`. This array is used in FAT operations of a lower level than `fat_part_mounted[]`. Specifically, when the device is being partitioned, formatted and/or enumerated. Calling `fat_AutoMount()` relieves most applications of the need to directly use `fat_device_table[]`.

---

---

## fat\_AutoMount (cont'd)

---

---

### PARAMETERS

#### flags

Run-time device configuration flags to allow overriding the default device configuration flags. If not overriding the default configuration flags, specify `FDDF_USE_DEFAULT`. To override the default flags, specify the ORed combination of one or more of the following:

- `FDDF_MOUNT_PART_0`: Mount specified partition
- `FDDF_MOUNT_PART_1`:
- `FDDF_MOUNT_PART_2`:
- `FDDF_MOUNT_PART_3`:
- `FDDF_MOUNT_PART_ALL`: Mount all partitions
- `FDDF_MOUNT_DEV_0`: Apply to specified device
- `FDDF_MOUNT_DEV_1`:
- `FDDF_MOUNT_DEV_2`:
- `FDDF_MOUNT_DEV_3`:
- `FDDF_MOUNT_DEV_ALL`: Apply to all available devices
- `FDDF_NO_RECOVERY`: Use norecovery if fails first time
- `FDDF_COND_DEV_FORMAT`: Format device if unformatted
- `FDDF_COND_PART_FORMAT`: Format partition if unformatted
- `FDDF_UNCOND_DEV_FORMAT`: Format device unconditionally
- `FDDF_UNCOND_PART_FORMAT`: Format partition unconditionally

**Note:** The `FDDF_MOUNT_PART_*` flags apply equally to all `FDDF_MOUNT_DEV_*` devices which are specified. If this is a problem, call this function multiple times with a single `DEV` flag bit each time.

**Note:** Formatting the device creates a single FAT partition covering the entire device. It is recommended that you always set the `*_PART_FORMAT` flag bit if you set the corresponding `*_DEV_FORMAT` flag bit.

---

---

## fat\_AutoMount (cont'd)

---

---

### RETURN VALUE

- 0: success
- EBADPART: partition is not a valid FAT partition
- EIO: Device I/O error
- EINVAL: invalid prtTable
- EUNFORMAT: device is not formatted
- ENOPART: no partitions exist on the device
- EBUSY: For non-blocking mode only, the device is busy. Call this function again to complete the close.

Any other negative value means that an I/O error occurred when updating the directory entry. In this case, the file is forced to close, but its recorded length might not be valid.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_EnumDevice, fat\_EnumPartition, fat\_MountPartition

---

---

## fat\_Close

---

---

```
int fat_Close( FATfile *file );
```

### DESCRIPTION

Closes a currently open file. You should check the return code since an I/O needs to be performed when closing a file to update the file's EOF offset (length), last access date, attributes and last write date (if modified) in the directory entry. This is particularly critical when using non-blocking mode.

### PARAMETERS

**file**                    Pointer to the open file to close.

### RETURN VALUE

0: success.  
-EINVAL: invalid file handle.  
-EBUSY: For non-blocking mode only, the device is busy. Call this function again to complete the close.

Any other negative value means that an I/O error occurred when updating the directory entry. In this case, the file is forced to close, but its recorded length might not be valid.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_Open, fat\_OpenDir

---

---

## fat\_CreateDir

---

---

```
int fat_CreateDir( fat_part *part, char *dirname );
```

### DESCRIPTION

Creates a directory if it does not already exist. The parent directory must already exist.

In non-blocking mode, only one file or directory can be created at any one time, since a single static `FATfile` is used for temporary storage. Each time you call this function, pass the same `dirname` pointer (not just the same string contents).

### PARAMETERS

<b>part</b>	Handle for the partition being used.
<b>dirname</b>	Pointer to the full path name of the directory to be created.

### RETURN VALUE

0: success.

- EINVAL: invalid argument. Trying to create volume label.
- ENOENT: parent directory does not exist.
- EPERM: the directory already exists or is write-protected.
- EBUSY: the device is busy (only if non-blocking).
- EFSTATE: if non-blocking, but a previous sequence of calls to this function (or `fat_CreateFile()`) has not completed and you are trying to create a different file or directory. You must complete the sequence of calls for each file or directory i.e., keep calling until something other than `-EBUSY` is returned.

Other negative values are possible from `fat_Open()/fat_Close()` calls.

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_ReadDir`, `fat_Status`, `fat_Open`, `fat_CreateFile`

---

---

## fat\_CreateFile

---

---

```
int fat_CreateFile( fat_part *part, char *filename, long alloc_size,
    FATfile *file );
```

### DESCRIPTION

Creates a file if it does not already exist. The parent directory must already exist.

In non-blocking mode, if `file` is NULL, only one file or directory can be created at any one time, since a single static `FATfile` is used for temporary storage. Each time you call this function, pass the same `dirname` pointer (not just the same string contents).

### PARAMETERS

<b>part</b>	Pointer to the partition being used.
<b>filename</b>	Pointer to the full path name of the file to be created.
<b>alloc_size</b>	Initial number of bytes to pre-allocate. Note that at least one cluster will be allocated. If there is not enough space beyond the first cluster for the requested allocation amount, the file will be allocated with whatever space is available on the partition, but no error code will be returned. If not even the first cluster is allocated, the <code>-ENOSPC</code> error code will return. This initial allocation amount is rounded up to the next whole number of clusters.
<b>file</b>	If not NULL, the created file is opened and accessible using this handle. If NULL, the file is closed after it is created.

### RETURN VALUE

0: success.  
-EINVAL: `part`, `filename`, `alloc_size`, or `file` contain invalid values.  
-ENOENT: the parent directory does not exist.  
-ENOSPC: no allocatable sectors were found.  
-EPERM: write-protected, trying to create a file on a read-only partition.  
-EBUSY: the device is busy (non-blocking mode only).  
-EFSTATE: if non-blocking, but a previous sequence of calls to this function (of `fat_CreateFile`) has not completed but you are trying to create a different file or directory. You must complete the sequence of calls for each file or directory i.e. keep calling until something other than `-EBUSY` is returned. This code is only returned if you pass a NULL file pointer, or if the file pointer is not NULL and the referenced file is already open.

Other negative values indicate I/O error, etc.

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_ReadDir`, `fat_Write`

---

---

## fat\_CreateTime

---

---

```
int fat_CreateTime( fat_dirent *entry, struct tm *t );
```

### DESCRIPTION

This function puts the creation date and time of the entry into the system time structure `t`. The function does not fill in the `tm_wday` field in the system time structure.

### PARAMETERS

<code>entry</code>	Pointer to a directory entry
<code>t</code>	Pointer to a system time structure

### RETURN VALUE

0: success.  
-EINVAL: invalid directory entry or time pointer

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_ReadDir`, `fat_Status`, `fat_LastAccess`, `fat_LastWrite`

---

---

## fat\_Delete

---

---

```
int fat_Delete( fat_part *part, int type, char *name );
```

### DESCRIPTION

Deletes the specified file or directory. The `type` must match or the deletion will not occur. This routine inserts a deletion code into the directory entry and marks the sectors as available in the FAT table, but does not actually destroy the data contained in the sectors. This allows an undelete function to be implemented, but such a routine is not part of this library. A directory must be empty to be deleted.

### PARAMETERS

<b>part</b>	Handle for the partition being used.
<b>type</b>	Must be a FAT file ( <code>FAT_FILE</code> ) or a FAT directory ( <code>FAT_DIR</code> ), depending on what is to be deleted.
<b>name</b>	Pointer to the full path name of the file/directory to be deleted.

### RETURN VALUE

0: success.  
-EIO: device I/O error.  
-EINVAL: `part`, `type`, or `name` contain invalid values.  
-EPATHSTR: `name` is not a valid path/name string.  
-EPERM: the file is open, write-protected, hidden, or system.  
-ENOTEMPTY: the directory is not empty.  
-ENOENT: the file/directory does not exist.  
-EBUSY: the device is busy. (Only if non-blocking.)  
-EPSTATE: if the partition is busy; i.e., there is an allocation in progress. (Only if non-blocking.)

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_OpenDir`, `fat_Split`, `fat_Truncate`, `fat_Close`



---

---

## fat\_EnumDevice

---

---

```
int fat_EnumDevice( mbr_drvr *driver, mbr_dev *dev, int devnum,
    char *sig, int norecovery );
```

### DESCRIPTION

This routine is called to learn about the devices present on the driver passed in. The device will be added to the linked list of enumerated devices. Partition pointers will be set to NULL, indicating they have not been enumerated yet. Partition entries must be enumerated separately.

The signature string is an identifier given to the write-back cache, and must remain consistent between resets so that the device can be associated properly with any battery-backed cache entries remaining in memory.

This function is called by `fat_AutoMount()` and `fat_Init()`.

### PARAMETERS

<b>driver</b>	Pointer to an initialized driver structure set up during the initialization of the storage device driver.
<b>dev</b>	Pointer to the device structure to be filled in.
<b>devnum</b>	Physical device number of the device.
<b>sig</b>	Pointer to a unique signature string. Note that this value <b>must</b> remain the same between resets.
<b>norecovery</b>	Boolean flag - set to True to ignore power-recovery data. True is any value except zero.

### RETURN VALUE

- 0: success.
- EIO: error trying to read the device or structure.
- EINVAL: devnum invalid or does not exist.
- ENOMEM: memory for page buffer/RJ is not available.
- EUNFORMAT: the device is accessible, but not formatted. You may use it provided it is formatted/partitioned by either this library or by another system.
- EBADPART: the partition table on the device is invalid.
- ENOPART: the device does not have any FAT partitions. This code is superseded by any other error detected.
- EEXIST: the device has already been enumerated.
- EBUSY: the device is busy (nonblocking mode only).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_AutoMount`, `fat_Init`, `fat_EnumPartition`

---

---

## fat\_EnumPartition

---

---

```
int fat_EnumPartition( mbr_dev *dev, int pnum, fat_part *part );
```

### DESCRIPTION

This routine is called to enumerate a partition on the given device. The partition information will be put into the FAT partition structure pointed to by `part`. The partition pointer will be linked to the device structure, registered with the write-back cache, and will then be active. The partition must be of a valid FAT type.

This function is called by `fat_AutoMount()` and `fat_Init()`.

### PARAMETERS

<b>dev</b>	Pointer to an MBR device structure.
<b>pnum</b>	Partition number to link and enumerate.
<b>part</b>	Pointer to an FAT partition structure to be filled in.

### RETURN VALUE

0: success.

- EIO: error trying to read the device or structure.
- EINVAL: partition number is invalid.
- EUNFORMAT: the device is accessible, but not formatted.
- EBADPART: the partition is not a FAT partition.
- EEXIST: the partition has already been enumerated.
- EUNFLUSHABLE: there are no flushable sectors in the write-back cache.
- EBUSY: the device is busy (Only if non-blocking.).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_EnumDevice`, `fat_FormatPartition`, `fat_MountPartition`

---

---

## fat\_FileSize

---

---

```
int fat_FileSize( FATfile *file, unsigned long *length );
```

### DESCRIPTION

Puts the current size of the file in bytes into `length`.

### PARAMETERS

<code>file</code>	Handle for an open file.
<code>length</code>	Pointer to the variable where the file length (in bytes) is to be placed.

### RETURN VALUE

0: success.  
-EINVAL: `file` is invalid.

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_Seek`

---

---

## fat\_FormatDevice

---

---

```
int fat_FormatDevice( mbr_dev *dev, int mode );
```

### DESCRIPTION

Formats a device. The device will have a DOS master boot record (MBR) written to it. Existing partitions are left alone if the device was previously formatted. The formatted device will be registered with the write-back cache for use with the FAT library. The one partition mode will instruct the routine to create a partition table, with one partition using the entire device. This mode only works if the device is currently unformatted or has no partitions.

If needed (i.e., there is no MBR on the device), this function is called by `fat_AutoMount()` if its flags parameter allows it.

### PARAMETERS

<b>dev</b>	Pointer to the data structure for the device to format.
<b>mode</b>	Mode: 0 = normal (use the partition table in the device structure) 1 = one partition using the entire device (errors occur if there are already partitions in the device structure) 3 = force one partition for the entire device (overwrites values already in the device structure)

### RETURN

0: success.  
-EIO: error trying to read the device or structure.  
-EINVAL: device structure is invalid or does not exist.  
-ENOMEM: memory for page buffer/RJ is not available.  
-EEXIST: the device is already formatted.  
-EPERM: the device already has mounted partition(s).  
-EBUSY: the device is busy. (Only if non-blocking.)

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_AutoMount`, `fat_Init`, `fat_EnumDevice`, `fat_PartitionDevice`,  
`fat_FormatPartition`

---

---

## fat\_FormatPartition

---

---

```
int fat_FormatPartition( mbr_dev *dev, fat_part *part, int pnum,
                        int type, char *label, int (*usr)() );
```

### DESCRIPTION

Formats partition number `pnum` according to partition type. The partition table information in the device must be valid. This will always be the case if the device was enumerated. The partition type must be a valid FAT type. Also note that the partition is *not* mounted after the partition is formatted. If `-EBUSY` is returned, the partition structure must not be disturbed until a subsequent call returns something other than `-EBUSY`.

If needed (i.e., `fat_MountPartition()` returned error code `-EBADPART`), this function is called by `fat_AutoMount()`.

### PARAMETERS

<code>dev</code>	Pointer to a device structure containing partitions.
<code>part</code>	Pointer to a FAT partition structure to be linked. Note that <code>opstate</code> <i>must</i> be set to zero before first call to this function if the library is being used in the non-blocking mode.
<code>pnum</code>	Partition number on the device (0–3).
<code>type</code>	Partition type.
<code>label</code>	Pointer to a partition label string.
<code>usr</code>	Pointer to a user routine.

### RETURN VALUE

- 0: success.
- `-EIO`: error in reading the device or structure.
- `-EINVAL`: the partition number is invalid.
- `-EPERM`: write access is not allowed.
- `-EUNFORMAT`: the device is accessible, but is not formatted.
- `-EBADPART`: the partition is not a valid FAT partition.
- `-EACCES`: the partition is currently mounted.
- `-EBUSY`: the device is busy (Only if non-blocking.).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_AutoMount`, `fat_Init`, `fat_FormatDevice`, `fat_EnumDevice`,  
`fat_PartitionDevice`, `fat_EnumPartition`

---

---

## fat\_Free

---

---

```
int fat_Free( fat_part *part );
```

### DESCRIPTION

This function returns the number of free clusters on the partition.

### PARAMETERS

**part**                    Handle to the partition.

### RETURN VALUE

Number of free clusters on success  
0: partition handle is bad or partition is not mounted.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_EnumPartition, fat\_MountPartition

---

---

## fat\_GetAttr

---

---

```
int fat_GetAttr( FATfile *file );
```

### DESCRIPTION

This function gets the given attributes to the file. Use the defined attribute flags to check the value:

- FATATTR\_READ\_ONLY - The file can not be modified.
- FATATTR\_HIDDEN - The file is not visible when doing normal operations.
- FATATTR\_SYSTEM - This is a system file and should be left alone.
- FATATTR\_VOLUME\_ID - This is the name of a logical disk.
- FATATTR\_DIRECTORY - This is a directory and not a file.
- FATATTR\_ARCHIVE - This tells you when the file was last modified.
- FATATTR\_LONG\_NAME - This is a FAT32 or long file name. It is not supported.

### PARAMETERS

**file**                    Handle to the open file.

### RETURN VALUE

Attributes on success  
-EINVAL: invalid file handle.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_Open, fat\_Status

---

---

## fat\_GetName

---

---

```
int fat_GetName( fat_dirent *entry, char *buf, word flags );
```

### DESCRIPTION

Translates the file or directory name in the `fat_dirent` structure into a printable name. FAT file names are stored in a strict fixed-field format in the `fat_dirent` structure (returned from `fat_Status`, for example). This format is not always suitable for printing, so this function should be used to convert the name to a printable null-terminated string.

### PARAMETERS

<b>entry</b>	Pointer to a directory entry obtained by <code>fat_Status()</code> .
<b>buf</b>	Pointer to a <code>char</code> array that will be filled in. This array must be at least 13 characters long.
<b>flags</b>	May be one of the following: <ul style="list-style-type: none"><li>• 0 - standard format, e.g., <code>AUTOEXEC.BAT</code> or <code>XYZ.GIF</code></li><li>• <code>FAT_LOWERCASE</code> - standard format, but make lower case.</li></ul>

### RETURN VALUE

0: success.  
-EINVAL: invalid (NULL) parameter(s).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_ReadDir`, `fat_Status`



---

---

## fat\_Init

---

---

```
int fat_Init( int pnum, mbr_drvr *driver, mbr_dev *dev, fat_part
             *part, int norecovery );
```

### DESCRIPTION

Initializes the default driver in MBR\_DRIVER\_INIT, enumerates device 0, then enumerates and mounts the specified partition. This function was replaced with the more powerful `fat_AutoMount()`.

`fat_Init()` will only work with device 0 of the default driver. This driver becomes the primary driver in the system.

The application can start calling any directory or file functions after this routine returns successfully.

The desired partition must already be formatted. If the partition mount fails, you may call the function again using a different partition number (`pnum`). The device will not be initialized a second time.

### PARAMETERS

<b>pnum</b>	Partition number to mount (0-3).
<b>driver</b>	Pointer to the driver structure to fill in.
<b>dev</b>	Pointer to the device structure to fill in.
<b>part</b>	Pointer to the partition structure to fill in.
<b>norecovery</b>	Boolean flag - set to True to ignore power-recovery data. True is any value except zero.

### RETURN VALUE

- 0: success.
- EIO: device I/O error.
- EINVAL: `pnum`, `driver`, or `device`, or `part` is invalid.
- EUNFORMAT: the device is not formatted.
- EBADPART: the partition requested is not a valid FAT partition.
- ENOPART: no partitions exist on the device.
- EBUSY: the device is busy. (Only if non-blocking.)

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_AutoMount`, `fat_EnumDevice`, `fat_EnumPartition`,  
`fat_MountPartition`

---

---

## fat\_InitUCOSMutex

---

---

```
void fat_InitUCOSMutex( int mutexPriority );
```

### DESCRIPTION

This function was introduced in FAT version 2.10. Prior versions of the FAT file system are compatible with  $\mu$ C/OS-II only if FAT API calls are confined to one  $\mu$ C/OS-II task. The FAT API is not reentrant from multiple tasks without the changes made in FAT version 2.10. If you wish to use the FAT file system from multiple  $\mu$ C/COS tasks, you must do the following:

1. The statement `#define FAT_USE_UCOS_MUTEX` must come before the statement:  

```
#use FAT.LIB
```
2. After calling `OSInit()` and before starting any tasks that use the FAT, call `fat_InitUCOSMutex(mutexPriority)`. The parameter `mutexPriority` is a  $\mu$ C/OS-II task priority that *must* be higher than the priorities of all tasks that call FAT API functions.
3. You must not call low-level, non-API FAT or write-back cache functions. Only call FAT functions appended with “fat\_” and with public function descriptions.
4. Run the FAT in blocking mode (`#define FAT_BLOCK`).

Mutex timeouts or other errors will cause a run-time error `-ERR_FAT_MUTEX_ERROR`.

$\mu$ C/OS-II may raise the priority of tasks using mutexes to prevent priority inversion.

The default mutex time-out in seconds is given by `FAT_MUTEX_TIMEOUT_SEC`, which defaults to 5 seconds if not defined in the application before the statement `#use FAT.LIB`.

### PARAMETERS

**mutexPriority**     A  $\mu$ C/OS-II task priority that **MUST** be higher than the priorities of all tasks that call FAT API functions.

### RETURN VALUE

None: success.

`-ERR_FAT_MUTEX_ERROR`: A run-time error causes an exception and the application will exit with this error code.

### LIBRARY

`FAT.LIB`

### SEE ALSO

`fat_AutoMount`, `fat_Init`

---

---

## fat\_LastAccess

---

---

```
int fat_LastAccess( fat_dirent *entry, struct tm *t );
```

### DESCRIPTION

Puts the last access date of the specified entry into the system time structure `t`. The time is always set to midnight. The function does *not* fill in the `tm_wday` field in the system time structure.

### PARAMETERS

<code>entry</code>	Pointer to a directory entry
<code>t</code>	Pointer to a system time structure

### RETURN VALUE

0: success.  
-EINVAL: invalid directory entry or time pointer

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_ReadDir`, `fat_Status`, `fat_CreateTime`, `fat_LastWrite`

---

---

## fat\_LastWrite

---

---

```
int fat_LastWrite( fat_dirent *entry, struct tm *t );
```

### DESCRIPTION

Puts the date and time of the last write for the given entry into the system time structure `t`. The function does not fill in the `tm_wday` field in the system time structure.

### PARAMETERS

<code>entry</code>	Pointer to a directory entry
<code>t</code>	Pointer to a system time structure

### RETURN VALUE

0: success.  
-EINVAL: invalid directory entry or time pointer

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_ReadDir`, `fat_Status`, `fat_CreateTime`, `fat_LastAccess`

---

---

## fat\_MountPartition

---

---

```
int fat_MountPartition( fat_part *part );
```

### DESCRIPTION

Marks the enumerated partition as mounted on both the FAT and MBR level. The partition MUST be previously enumerated with `fat_EnumPartition()`.

This function is called by `fat_AutoMount()` and `fat_Init()`.

### PARAMETER

**part**                    Pointer to the FAT partition structure to mount.

### RETURN VALUE

- 0: success.
- EINVAL: device or partition structure or `part` is invalid.
- EBADPART: the partition is not a FAT partition.
- ENOPART: the partition does not exist on the device.
- EPERM: the partition has not been enumerated.
- EACCESS: the partition is already linked to another `fat_part` structure.
- EBUSY: the device is busy. (Only if non-blocking.)

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_EnumPartition`, `fat_UnmountPartition`

---

---

## fat\_Open

---

---

```
int fat_Open( fat_part *part, char *name, int type, int ff,
             FATfile *file, long *prealloc );
```

### DESCRIPTION

Opens a file or directory, optionally creating it if it does not already exist. If the function returns -EBUSY, call it repeatedly with the same arguments until it returns something other than -EBUSY.

### PARAMETERS

<b>part</b>	Handle for the partition being used.
<b>name</b>	Pointer to the full path name of the file to be opened/created.
<b>type</b>	FAT_FILE or FAT_DIR, depending on what is to be opened/created.
<b>ff</b>	File flags, must be one of: <ul style="list-style-type: none"><li>• FAT_OPEN - Object must already exist. If it does not exist, -ENOENT will be returned.</li><li>• FAT_CREATE - Object is created only if it does not already exist</li><li>• FAT_MUST_CREATE - Object is created, and it must not already exist.</li><li>• FAT_READONLY - No write operations (this flag is mutually exclusive with any of the CREATE flags).</li><li>• FAT_SEQUENTIAL - Optimize for sequential reads and/or writes. This setting can be changed while the file is open by using the <code>fat_fcntl()</code> function.</li></ul>
<b>file</b>	Pointer to an empty FAT file structure that will act as a handle for the newly opened file. Note that you must <code>memset</code> this structure to zero when you are using the non-blocking mode before calling this function the first time. Keep calling until something other than -EBUSY is returned, but do not change anything in any of the parameters while doing so.
<b>prealloc</b>	An initial byte count if the object needs to be created. This number is rounded up to the nearest whole number of clusters greater than or equal to 1. This parameter is only used if one of the *_CREATE flag is set and the object does not already exist. On return, *prealloc is updated to the actual number of bytes allocated. May be NULL, in which case one cluster is allocated if the call is successful.

---

---

## fat\_Open (cont'd)

---

---

### RETURN VALUE

0: success.

- EINVAL: invalid arguments. Trying to create volume label, or conflicting flags.
- ENOENT: file/directory could not be found.
- EEXIST: object existed when FAT\_MUST\_CREATE flag set.
- EPERM: trying to create a file/directory on a read-only partition.
- EMFILE - too many open files. If you get this code, increase the FAT\_MAXMARKERS definition in the BIOS.

Other negative values indicate I/O error, etc.

Non-blocking mode only:

- EBUSY: the device is busy (nonblocking mode only).
- EFSTATE - file structure is not in a valid state. Usually means it was not zeroed before calling this function for the first time (for that file) struct, when in non-blocking mode; can also occur if the same file struct is opened more than once.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_ReadDir, fat\_Status, fat\_Close

---

---

## fat\_OpenDir

---

---

```
int fat_OpenDir( fat_part *part, char *dirname, FATfile *dir );
```

### DESCRIPTION

Opens a directory for use, filling in the FATfile handle.

### PARAMETERS

<b>part</b>	Pointer to the partition structure being used.
<b>dirname</b>	Pointer to the full path name of the directory to be opened or created.
<b>dir</b>	Pointer to directory requested.

### RETURN VALUE

0: success  
-EINVAL: invalid argument.  
-ENOENT: the directory cannot be found.  
-EBUSY: the device is busy (Only if non-blocking).

Other negative values are possible from the fat\_Open() call.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_ReadDir, fat\_Status, fat\_Open, fat\_Close



---

---

## fat\_PartitionDevice

---

---

```
int fat_PartitionDevice( mbr_dev *dev, int pnum );
```

### DESCRIPTION

This function partitions the device by modifying the master boot record (MBR), which could destroy access to information already on the device. The partition information contained in the specified `mbr_dev` structure must be meaningful, and the sizes and start positions must make sense (no overlapping, etc.). If this is not true, you will get an `-EINVAL` error code. The device being partitioned must already have been formatted and enumerated.

This function will only allow changes to one partition at a time, and this partition must either not exist or be of a FAT type.

The validity of the new partition will be verified before any changes are done to the device. All other partition information in the device structure (for those partitions that are not being modified) must match the values currently existing on the MBR. The type given for the new partition must either be zero (if you are deleting the partition) or a FAT type.

You may not use this function to create or modify a non-FAT partition.

### PARAMETERS

<b>dev</b>	Pointer to the device structure of the device to be partitioned.
<b>pnum</b>	Partition number of the partition being modified.

### RETURN VALUE

0: success.  
-EIO: device I/O error.  
-EINVAL: `pnum` or device structure is invalid.  
-EUNFORMAT: the device is not formatted.  
-EBADPART: the partition is a non-FAT partition.  
-EPERM: the partition is mounted.  
-EBUSY: the device is busy (Only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_FormatDevice`, `fat_EnumDevice`, `fat_FormatPartition`

---

---

## fat\_Read

---

---

```
int fat_Read( FATfile *file, char *buf, int len );
```

### DESCRIPTION

Given `file`, `buf`, and `len`, this routine reads `len` characters from the specified file and places the characters into `buf`. The function returns the number of characters actually read on success. Characters are read beginning at the current position of the file and the position pointer will be left pointing to the next byte to be read. The file position can be changed by the `fat_Seek()` function. If the file contains fewer than `len` characters from the current position to the EOF, the transfer will stop at the EOF. If already at the EOF, 0 is returned. The `len` parameter must be positive, limiting reads to 32767 bytes per call.

### PARAMETERS

<b>file</b>	Handle for the file being read.
<b>buf</b>	Pointer to the buffer where data are to be placed.
<b>len</b>	Length of data to be read.

### RETURN VALUE

Number of bytes read: success. May be less than the requested amount in non-blocking mode, or if EOF was encountered.

- EEOF: starting position for read was at (or beyond) end-of-file.
- EIO: device I/O error.
- EINVAL: `file`, `buf`, or `len`, contain invalid values.
- EPERM: the file is locked.
- ENOENT: the file/directory does not exist.
- EFSTATE: file is in inappropriate state (Only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_Write`, `fat_Seek`

---

---

## fat\_ReadDir

---

---

```
int fat_ReadDir( FATfile *dir, fat_dirent *entry, int mode );
```

### DESCRIPTION

Reads the next entry of the desired type from the given directory, filling in the entry structure.

### PARAMETERS

**dir** Pointer to the handle for the directory being read.

**entry** Pointer to the handle to the entry structure to fill in.

**mode** 0 = next active file or directory entry including read only (no hidden, sys, label, deleted or empty)

A nonzero value sets the selection based on the following attributes:

- FATATTR\_READ\_ONLY - include read-only entries
- FATATTR\_HIDDEN - include hidden entries
- FATATTR\_SYSTEM - include system entries
- FATATTR\_VOLUME\_ID - include label entries
- FATATTR\_DIRECTORY - include directory entries
- FATATTR\_ARCHIVE - include modified entries
- FAT\_FIL\_RD\_ONLY - filter on read-only attribute
- FAT\_FIL\_HIDDEN - filter on hidden attribute
- FAT\_FIL\_SYSTEM - filter on system attribute
- FAT\_FIL\_LABEL - filter on label attribute
- FAT\_FIL\_DIR - filter on directory attribute
- FAT\_FIL\_ARCHIVE - filter on modified attribute

The FAT\_INC\_\* flags default to FAT\_INC\_ACTIVE if none set:

- FAT\_INC\_DELETED - include deleted entries
- FAT\_INC\_EMPTY - include empty entries
- FAT\_INC\_LNAME - include long name entries
- FAT\_INC\_ACTIVE - include active entries

The following predefined filters are available:

- FAT\_INC\_ALL - returns ALL entries of ANY type
- FAT\_INC\_DEF - default (files and directories including read-only and archive)

**Note:** Active files are included by default unless FAT\_INC\_DELETED, FAT\_INC\_EMPTY, or FAT\_INC\_LNAME is set. Include flags become the desired filter value if the associated filter flags are set.

---

---

## fat\_ReadDir (cont'd)

---

---

### EXAMPLES OF FILTER BEHAVIOR

mode = FAT\_INC\_DEF | FATFIL\_HIDDEN | FATATTR\_HIDDEN

would return the next hidden file or directory (including read-only and archive)

mode = FAT\_INC\_DEF | FAT\_FIL\_HIDDEN | FAT\_FIL\_DIR | FATATTR\_HIDDEN

would return next hidden directory (but would not return any hidden file)

mode = FAT\_INC\_DEF | FAT\_FIL\_HIDDEN | FAT\_FIL\_DIR |  
FATATTR\_HIDDEN & ~FATATTR\_DIRECTORY

would return next hidden file (but would not return any hidden directory)

mode = FAT\_INC\_ALL & ~FAT\_INC\_EMPTY

would return the next non-empty entry of any type

### RETURN VALUE

0: success.

- EINVAL: invalid argument.
- ENOENT: directory does not exist
- EEOF: no more entries in the directory
- EFAULT: directory chain has link error
- EBUSY: the device is busy (non-blocking mode only)

Other negative values from the `fat_Open()` call are also possible.

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_OpenDir`, `fat_Status`

---

---

## fat\_Seek

---

---

```
int fat_Seek( FATfile *file, long pos, int whence );
```

### DESCRIPTION

Positions the internal file position pointer. `fat_Seek()` will allocate clusters to the file if necessary, but will not move the position pointer beyond the original end of file (EOF) unless doing a `SEEK_RAW`. In all other cases, extending the pointer past the original EOF will preallocate the space that would be needed to position the pointer as requested, but the pointer will be left at the original EOF and the file length will not be changed. If this occurs, an EOF error will be returned to indicate the space was allocated but the pointer was left at the EOF.

### PARAMETERS

- |               |  |
|---------------|--|
| <b>file</b>   | Pointer to the file structure of the open file.  |
| <b>pos</b>    | Position value in number of bytes (may be negative). This value is interpreted according to the third parameter, <i>whence</i> .   |
| <b>whence</b> | Must be one of the following: <ul style="list-style-type: none"><li>• <code>SEEK_SET</code> - <code>pos</code> is the byte position to seek, where 0 is the first byte of the file. If <code>pos</code> is less than 0, the position pointer is set to 0 and no error code is returned. If <code>pos</code> is greater than the length of the file, the position pointer is set to EOF and error code <code>-EEOF</code> is returned.</li><li>• <code>SEEK_CUR</code> - seek <code>pos</code> bytes from the current position. If <code>pos</code> is less than 0 the seek is towards the start of the file. If this goes past the start of the file, the position pointer is set to 0 and no error code is returned. If <code>pos</code> is greater than 0 the seek is towards EOF. If this goes past EOF the position pointer is set to EOF and error code <code>-EEOF</code> is returned.</li><li>• <code>SEEK_END</code> - seek to <code>pos</code> bytes from the end of the file. That is, for a file that is <code>x</code> bytes long, the statement:<br/><pre>fat_Seek (&amp;my_file, -1, SEEK_END);</pre>will cause the position pointer to be set at <code>x-1</code> no matter its value prior to the seek call. If the value of <code>pos</code> would move the position pointer past the start of the file, the position pointer is set to 0 (the start of the file) and no error code is returned. If <code>pos</code> is greater than or equal to 0, the position pointer is set to EOF and error code <code>-EEOF</code> is returned..</li><li>• <code>SEEK_RAW</code> - is similar to <code>SEEK_SET</code>, but if <code>pos</code> goes beyond EOF, using <code>SEEK_RAW</code> will set the file length and the position pointer to <code>pos</code>.</li></ul> |

---

---

## fat\_Seek (cont'd)

---

---

### RETURN VALUE

- 0: success.
- EIO: device I/O error.
- EINVAL: file, pos, or whence contain invalid values.
- EPERM: the file is locked or writes are not permitted.
- ENOENT: the file does not exist.
- EEOF: space is allocated, but the pointer is left at original EOF.
- ENOSPC: no space is left on the device to complete the seek.
- EBUSY: the device is busy (Only if non-blocking).
- EFSTATE: if file in inappropriate state (Only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_Open, fat\_Read, fat\_Write, fat\_xWrite

---

---

## fat\_SetAttr

---

---

```
int fat_SetAttr( FATfile *file, int attr );
```

### DESCRIPTION

This function sets the given attributes to the file. Use defined attribute flags to create the set values.

### PARAMETERS

<b>file</b>	Handle to the open file.
<b>attr</b>	Attributes to set in file. For attribute description see <a href="#">fat_GetAttr()</a> . May be one or more of the following: <ul style="list-style-type: none"><li>• FATATTR_READ_ONLY</li><li>• FATATTR_HIDDEN</li><li>• FATATTR_SYSTEM</li><li>• FATATTR_VOLUME_ID</li><li>• FATATTR_DIRECTORY</li><li>• FATATTR_ARCHIVE</li><li>• FATATTR_LONG_NAME</li></ul>

### RETURN VALUE

0: Success  
-EIO: on device IO error  
-EINVAL: invalid open file handle  
-EPERM: if the file is locked or write not permitted  
-EBUSY: if the device is busy. (Only if non-blocking)

### LIBRARY

FAT.LIB

### SEE ALSO

[fat\\_Open](#), [fat\\_Status](#)

---

---

## fat\_Split

---

---

```
int fat_Split( FATfile *file, long where, char *newfile );
```

### DESCRIPTION

Splits the original file at `where` and assigns any left over allocated clusters to `newfile`. As the name implies, `newfile` is a newly created file that must not already exist. Upon completion, the original file is closed and the file handle is returned pointing to the created and opened new file. The file handle given must point to a file of type `FAT_FILE`. There are internal static variables used in this function, so only one file split operation can be active. Additional requests will be held off with `-EBUSY` returns until the active split completes.

### PARAMETERS

<b>file</b>	Pointer to the open file to split.
<b>where</b>	May be one of the following: <ul style="list-style-type: none"><li>• <math>\geq 0</math> - absolute byte to split the file. If the absolute byte is beyond the EOF, file is split at EOF.</li><li>• <code>FAT_BRK_END</code> - split at EOF.</li><li>• <code>FAT_BRK_POS</code> - split at current file position.</li></ul>
<b>newfile</b>	Pointer to the absolute path and name of the new file created for the split.

### RETURN VALUE

0: success.  
-EIO: device I/O error.  
-EINVAL: `file` has invalid references.  
-EPATHSTR: `newfile` is not a valid path/name string.  
-EEOF: no unused clusters are available for `newfile`. `file` will be unchanged and open, `newfile` is not created.  
-EPERM: `file` is in use, write-protected, hidden, or system.  
-ENOENT: `file` does not exist.  
-ETYPE: `file` is not a FAT file type.  
-EBUSY: the device is busy (Only non-blocking mode).  
-EFSTATE: if file in inappropriate state (Only non-blocking mode).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_OpenDir`, `fat_Delete`, `fat_Truncate`, `fat_Close`



---

---

## fat\_Status

---

---

```
int fat_Status( fat_part *part, char *name, fat_dirent *entry );
```

### DESCRIPTION

Scans for the specified entry and fills in the entry structure if found without opening the directory or entry.

### PARAMETERS

<b>part</b>	Pointer to the partition structure being used.
<b>name</b>	Pointer to the full path name of the entry to be found.
<b>entry</b>	Pointer to the directory entry structure to fill in.

### RETURN VALUE

0: success.  
-EIO: device I/O error.  
-EINVAL: part, filepath, or entry are invalid.  
-ENOENT: the file/directory/label does not exist.  
-EBUSY: the device is busy (Only non-blocking mode). If you get this error, call the function again without changing any parameters.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_ReadDir

---

---

## fat\_SyncFile

---

---

```
int fat_SyncFile( FATfile *file );
```

### DESCRIPTION

Updates the directory entry for the given file, committing cached size, dates, and attribute fields to the actual directory. This function has the same effect as closing and re-opening the file.

### PARAMETERS

**file**                    Pointer to the open file.

### RETURN VALUE

0: success.  
-EINVAL: `file` is invalid.  
-EPERM - this operation is not permitted on the root directory.  
-EBUSY: the device is busy (Only if non-blocking). Call function again to complete the update.  
-EFSTATE - file not open or in an invalid state.

Any other negative value: I/O error when updating the directory entry.

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Close`, `fat_Open`, `fat_OpenDir`

---

---

## fat\_SyncPartition

---

---

```
int fat_SyncPartition( fat_part *part );
```

### DESCRIPTION

Flushes all cached writes to the specified partition to the actual device.

### PARAMETER

**part**                    Pointer to the partition to be synchronized.

### RETURN VALUE

0: success.

-EINVAL: part is invalid.

-EBUSY: the device is busy (Only if non-blocking). Call function again to complete the sync.

Any other negative value: I/O error when updating the device.

### LIBRARY

FAT.LIB

### SEE ALSO

fat\_Close, fat\_SyncFile, fat\_UnmountPartition

---

---

## fat\_Tell

---

---

```
int fat_Tell( FATfile *file, unsigned long *pos );
```

### DESCRIPTION

Puts the value of the position pointer (that is, the number of bytes from the beginning of the file) into `pos`. Zero indicates the position pointer is at the beginning of the file.

### μC/OS-II USERS:

- The FAT API is not reentrant. To use the FAT from multiple μC/OS-II tasks, put the following statement in your application:

```
#define FAT_USE_UCOS_MUTEX
```
- Mutex timeouts or other mutex errors will cause the run-time error `ERR_FAT_MUTEX_ERROR`. The default mutex timeout is 5 seconds and can be changed by #define'ing a different value for `FAT_MUTEX_TIMEOUT_SEC`.
- You MUST call `fat_InitUCOSMutex()` after calling `OSInit()` and before calling any other FAT API functions.
- You must run the FAT in blocking mode (`#define FAT_BLOCK`).
- You must not call low-level, non-API FAT or write-back cache functions. Only call FAT functions appended with “`fat_`” and with public function descriptions.

### PARAMETERS

<b>file</b>	Pointer to the file structure of the open file
<b>pos</b>	Pointer to the variable where the value of the file position pointer is to be placed.

### RETURN VALUE

0: success.  
-EIO: position is beyond EOF.  
-EINVAL: file is invalid.

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Seek`, `fat_Read`, `fat_Write`, `fat_xWrite`

---

---

## fat\_tick

---

---

```
int fat_tick( void );
```

### DESCRIPTION

Drive device I/O completion and periodic flushing. It is not generally necessary for the application to call this function; however, if it is called regularly (when the application has nothing else to do) then file system performance may be improved.

### RETURN VALUE

Currently always 0.

### LIBRARY

FATWTC.LIB

---

---

## fat\_Truncate

---

---

```
int fat_Truncate( FATfile *file, long where );
```

### DESCRIPTION

Truncates the file at `where` and frees any left over allocated clusters. The file must be a `FAT_FILE` type.

### PARAMETERS

<b>file</b>	Pointer to the open file to truncate.
<b>where</b>	One of the following: <ul style="list-style-type: none"><li>• <math>\geq 0</math> - absolute byte to truncate the file. The file is truncated at EOF if the absolute byte is beyond EOF.</li><li>• <code>FAT_BRK_END</code> - truncate at EOF.</li><li>• <code>FAT_BRK_POS</code> - truncate at current file position.</li></ul>

### RETURN VALUE

0: success.  
-EIO: device I/O error.  
-EINVAL: `file` is invalid.  
-EPERM: `file` is in use, write-protected, hidden, or system.  
-ENOENT: the file does not exist.  
-ETYPE: `file` is not a FAT file type.  
-EBUSY: the device is busy (Only if non-blocking).  
-EFSTATE: if file in inappropriate state (Only if non-blocking)

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_OpenDir`, `fat_Delete`, `fat_Split`

---

---

## fat\_UnmountDevice

---

---

```
int fat_UnmountDevice( mbr_dev * dev );
```

### DESCRIPTION

Unmounts all FAT partitions on the given device and unregisters the device from the cache system. This commits all cache entries to the device and prepares the device for power down or removal. The device structure given must have been enumerated with `fat_EnumDevice()`.

This function was introduced in FAT module version 2.06. Applications using prior versions of the FAT module would call `fat_UnmountPartition()` instead.

### PARAMETER

**dev**                      Pointer to a FAT device structure to unmount.

### RETURN VALUE

- 0: success.
- EINVAL: device structure (`dev`) is invalid.
- EBUSY: the device is busy (Only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_EnumDevice`, `fat_AutoMount`, `fat_UnmountPartition`

---

---

## fat\_UnmountPartition

---

---

```
int fat_UnmountPartition( fat_part *part );
```

### DESCRIPTION

Marks the enumerated partition as unmounted on both the FAT and the master boot record levels. The partition must have been already enumerated using `fat_EnumPartition()` (which happens when you call `fat_AutoMount()`).

To unmount all FAT partitions on a device call `fat_UnmountDevice()`, a function introduced with FAT version 2.06. It not only commits all cache entries to the device, but also prepares the device for power down or removal.

**Note:** The partitions on a removable device must be unmounted in order to flush data before removal. Failure to unmount a partition that has been written could cause damage to the FAT file system.

### PARAMETERS

**part**                    Pointer to a FAT partition structure to unmount.

### RETURN VALUE

0: success.  
-EINVAL: device or partition structure or pnum is invalid.  
-EBADPART: the partition is not a FAT partition.  
-ENOPART: the partition does not exist on the device.  
-EPERM: the partition has not been enumerated.  
-EBUSY: the device is busy (only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_EnumPartition`, `fat_MountPartition`, `fat_UnmountDevice`



---

---

## fat\_Write

---

---

```
int fat_Write( FATfile *file, char *buf, int len );
```

### DESCRIPTION

Writes characters into the file specified by the file pointer beginning at the current position in the file. Characters will be copied from the string pointed to by `buf`. The `len` variable controls how many characters will be written. This can be more than one sector in length, and the write function will allocate additional sectors if needed. Data is written into the file starting at the current file position regardless of existing data. Overwriting at specific points in the file can be accomplished by calling the `fat_Seek()` function before calling `fat_Write()`.

### PARAMETERS

<code>file</code>	Handle for the open file being written.
<code>buf</code>	Pointer to the buffer containing data to write.
<code>len</code>	Length of data to be written.

### RETURN VALUE

Number of bytes written: success (may be less than `len`, or zero if non-blocking mode)

- EIO: device I/O error.
- EINVAL: `file`, `buf`, or `len` contain invalid values.
- ENOENT: file does not exist.
- ENOSPC: no space left on the device to complete the write.
- EFAULT: problem in file (broken cluster chain, etc.).
- EPERM: the file is locked or is write-protected.
- EBUSY: the device is busy (only if non-blocking).
- EFSTATE: file is in inappropriate state (only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_Read`, `fat_xWrite`, `fat_Seek`

---

---

## fat\_xWrite

---

---

```
int fat_xWrite( FATfile *file, long xbuf, int len );
```

### DESCRIPTION

Writes characters into the file specified by the file pointer beginning at the current position in the file. Characters will be copied from the `xmem` string pointed to by `xbuf`. The `len` variable controls how many characters will be written. This can be more than one sector in length, and the write function will allocate additional sectors if needed. Data will be written into the file starting at the current file position regardless of existing data. Overwriting at specific points in the file can be accomplished by calling the `fat_Seek()` function before calling `fat_xWrite()`.

### PARAMETERS

<code>file</code>	Handle for the open file being written.
<code>xbuf</code>	<code>xmem</code> address of the buffer to be written.
<code>len</code>	Length of data to write.

### RETURN VALUE

Number of bytes written: success. (may be less than `len`, or zero if non-blocking mode)

- EIO: device I/O error.
- EINVAL: `file`, `xbuf`, or `len` contain invalid values.
- ENOENT: the file/directory does not exist.
- ENOSPC: there are no more sectors to allocate on the device.
- EFAULT: there is a problem in the file (broken cluster chain, etc.).
- EPERM: the file is locked or write-protected.
- EBUSY: the device is busy (only if non-blocking).
- EFSTATE: file is in inappropriate state (only if non-blocking).

### LIBRARY

FAT.LIB

### SEE ALSO

`fat_Open`, `fat_Read`, `fat_Write`, `fat_Seek`

---

---

## fclose

---

---

```
void fclose( File* f );
```

### DESCRIPTION

Closes a file.

### PARAMETERS

**f**                      The pointer to the file to close.

### LIBRARY

FILESYSTEM.LIB

---

---

## fcreate (FS1)

---

---

```
int fcreate( File* f, FileNumber fnum );
```

### DESCRIPTION

Creates a file. Before calling this function, a variable of type `File` must be defined in the application program.

```
File file;  
fcreate (&file, 1);
```

### PARAMETERS

<b>f</b>	The pointer to the created file.
<b>fnum</b>	This is a user-defined number in the range of 1 to 127 inclusive. Each file in the flash file system is assigned a unique number in this range.

### RETURN VALUE

0: Success.  
1: Failure.

### LIBRARY

FILESYSTEM.LIB

---

---

## fcreate (FS2)

---

---

```
int fcreate( File* f, FileNumber name );
```

### DESCRIPTION

Create a new file with the given “file name” which is composed of two parts: the low byte is the actual file number (1 to 255 inclusive), and the high byte contains an extent number (1 to `_fs.num_lx`) on which to place the file metadata. The extent specified by `_fs.set_lx()` is always used to determine the actual data extent. If the high byte contains 0, then the default metadata extent specified by `_fs.set_lx()` is used. The file descriptor is filled in if successful. The file will be opened for writing, so a further call to `fopen_wr()` is not necessary.

The number of files which may be created is limited by the lower of `FS_MAX_FILES` and 255. This limit applies to the entire filesystem (all logical extents). Once a file is created, its data and metadata extent numbers are fixed for the life of the file, i.e., until the file is deleted.

When created, no space is allocated in the file system until the first write occurs for the file. Thus, if the system power is cycled after creation but before the first byte is written, the file will be effectively deleted. The first write to a file causes one sector to be allocated for the metadata.

Before calling this function, a variable of type `File` must be defined in the application program. (The `sizeof()` function will return the number of bytes used for the `File` data structure.)

```
File file;
fcreate (&file, 1);
```

### PARAMETERS

<b>f</b>	Pointer to the file descriptor to fill in.
<b>name</b>	File number including optional metadata extent number.

### RETURN VALUE

0: Success.  
!0: Failure.

### ERRNO VALUES

`EINVAL` - Zero file number requested, or invalid extent number.  
`EEXIST` - File with given number already exists.  
`ENFILE` - No space is available in the existing file table. If this error occurs, increase the definition of `FS_MAX_FILES`, a `#define` constant that should be declared before `#use "fs2.lib"`.

### LIBRARY

`fs2.LIB`

### SEE ALSO

`fcreate_unused (FS2)`, `_fs.set_lx (FS2)`, `fdelete (FS2)`

---

---

## `fcreate_unused (FS1)`

---

---

```
FileNumber fcreate_unused( File * f );
```

### DESCRIPTION

Searches for the first unused file number in the range 1 through 127, and creates a file with that number.

### PARAMETERS

`f`                    The pointer to the created file.

### RETURN VALUE

The FileNumber (1-127) of the new file if success.

### LIBRARY

FILESYSTEM.LIB

### SEE ALSO

`fcreate (FS1)`

---

---

## `fcreate_unused (FS2)`

---

---

```
FileNumber fcreate_unused( File * f );
```

### DESCRIPTION

Create a new file and return the “file name” which is a number between 1 and 255. The new file will be created on the current default extent(s) as specified by `fs_set_lx()`. Other behavior is the same as `fcreate()`.

### PARAMETERS

`f`                      Pointer to file descriptor to fill in.

### RETURN VALUE

>0: Success, the FileNumber (1-255) of the new file.  
0: Failure.

### ERRNO VALUE

ENFILE - No unused file number available.

### LIBRARY

`fs2.LIB`

### SEE ALSO

`fcreate (FS2)`, `fs_set_lx (FS2)`, `fdelete (FS2)`

---

---

## **fdelete (FS1)**

---

---

```
int fdelete( FileNumber fnum );
```

### **DESCRIPTION**

Deletes a file.

### **PARAMETERS**

<b>fnum</b>	A number in the range 1 to 127 inclusive that identifies the file in the flash file system.
-------------	---

### **RETURN VALUE**

0: Success.  
1: Failure.

### **LIBRARY**

FILESYSTEM.LIB



---

---

## **fdelete (FS2)**

---

---

```
int fdelete( FileNumber name );
```

### **DESCRIPTION**

Delete the file with the given number. The specified file must not be open. The file number (i.e. name) is composed of two parts: the low byte contains the actual file number, and the high byte (if not zero) contains the metadata extent number of the file.

### **PARAMETERS**

<b>name</b>	File number (1 to 255 inclusive).
-------------	-----------------------------------

### **RETURN VALUE**

0: Success.  
!0: Failure.

### **LIBRARY**

fs2.LIB

### **ERRNO VALUES**

ENOENT - File doesn't exist, or metadata extent number doesn't match an existing file.  
EBUSY - File is open.  
EIO - I/O error when releasing blocks occupied by this file.

### **SEE ALSO**

fcreate (FS2)

---

---

## **fflush (FS2)**

---

---

```
int fflush( File * f );
```

### **DESCRIPTION**

Flush any buffers, associated with the given file, retained in RAM to the underlying hardware device. This ensures that the file is completely written to the filesystem. The file system does not currently perform any buffering, however future revisions of this library may introduce buffering to improve performance.

### **PARAMETERS**

**f**                    Pointer to open file descriptor.

### **RETURN VALUE**

0: Success.  
!0: Failure.

### **ERRNO VALUES**

EBADFD - file invalid or not open.  
EIO - I/O error.

### **LIBRARY**

fs2.LIB

### **SEE ALSO**

fs\_sync (FS2)

---

---

## fftcplx

---

---

```
void fftcplx( int * x, int N, int * blockexp );
```

### DESCRIPTION

Computes the complex DFT of the N-point complex sequence contained in the array *x* and returns the complex result in *x*. *N* must be a power of 2 and lie between 4 and 1024. An invalid *N* causes a RANGE exception. The N-point complex sequence in array *x* is replaced with its N-point complex spectrum. The value of *blockexp* is increased by 1 each time array *x* has to be scaled, to avoid arithmetic overflow.

### PARAMETERS

<b>x</b>	Pointer to N-element array of complex fractions.
<b>N</b>	Number of complex elements in array <i>x</i> .
<b>blockexp</b>	Pointer to integer block exponent.

### LIBRARY

FFT.LIB

### SEE ALSO

`fftcplxinv`, `fftreal`, `fftrealinv`, `hanncplx`, `hannreal`, `powerspectrum`

---

---

## fftcplxinv

---

---

```
void fftcplxinv( int * x, int N, int * blockexp );
```

### DESCRIPTION

Computes the inverse complex DFT of the  $N$ -point complex spectrum contained in the array  $x$  and returns the complex result in  $x$ .  $N$  must be a power of 2 and lie between 4 and 1024. An invalid  $N$  causes a RANGE exception. The value of `blockexp` is increased by 1 each time array  $x$  has to be scaled, to avoid arithmetic overflow. The value of `blockexp` is also *decreased* by  $\log_2 N$  to include the  $1/N$  factor in the definition of the inverse DFT

### PARAMETERS

<b>x</b>	Pointer to $N$ -element array of complex fractions.
<b>N</b>	Number of complex elements in array $x$ .
<b>blockexp</b>	Pointer to integer block exponent.

### LIBRARY

FFT.LIB

### SEE ALSO

`fftcplx`, `fftreal`, `fftrealinv`, `hanncplx`, `hannreal`, `powerspectrum`

---

---

## fftreal

---

---

```
void fftreal( int * x, int N, int * blockexp );
```

### DESCRIPTION

Computes the  $N$ -point, positive-frequency complex spectrum of the  $2N$ -point real sequence in array  $x$ . The  $2N$ -point real sequence in array  $x$  is replaced with its  $N$ -point positive-frequency complex spectrum. The value of `blockexp` is increased by 1 each time array  $x$  has to be scaled, to avoid arithmetic overflow.

The imaginary part of the  $X[0]$  term (stored in  $x[1]$ ) is set to the real part of the  $fmax$  term.

The  $2N$ -point real sequence is stored in natural order. The zeroth element of the sequence is stored in  $x[0]$ , the first element in  $x[1]$ , and the  $k$ th element in  $x[k]$ .

$N$  must be a power of 2 and lie between 4 and 1024. An invalid  $N$  causes a RANGE exception.

### PARAMETERS

<b>x</b>	Pointer to $2N$ -point sequence of real fractions.
<b>N</b>	Number of complex elements in output spectrum
<b>blockexp</b>	Pointer to integer block exponent.

### LIBRARY

FFT.LIB

### SEE ALSO

`fftcplx`, `fftcplxinv`, `fftrealignv`, `hanncplx`, `hannreal`, `powerspectrum`

---

---

## fftreainv

---

---

```
void fftrealinv( int * x, int N, int * blockexp );
```

### DESCRIPTION

Computes the  $2N$ -point real sequence corresponding to the  $N$ -point, positive-frequency complex spectrum in array  $x$ . The  $N$ -point, positive-frequency spectrum contained in array  $x$  is replaced with its corresponding  $2N$ -point real sequence. The value of `blockexp` is increased by 1 each time array  $x$  has to be scaled, to avoid arithmetic overflow. The value of `blockexp` is also *decreased* by  $\log_2 N$  to include the  $1/N$  factor in the definition of the inverse DFT.

The function expects to find the real part of the  $f_{max}$  term in the imaginary part of the zero-frequency  $X[0]$  term (stored  $x[1]$ ).

The  $2N$ -point real sequence is stored in natural order. The zeroth element of the sequence is stored in  $x[0]$ , the first element in  $x[1]$ , and the  $k$ th element in  $x[k]$ .

$N$  must be a power of 2 and between 4 and 1024. An invalid  $N$  causes a RANGE exception.

### PARAMETERS

<b>x</b>	Pointer to $N$ -element array of complex fractions.
<b>N</b>	Number of complex elements in array $x$ .
<b>blockexp</b>	Pointer to integer block exponent.

### LIBRARY

FFT.LIB

### SEE ALSO

`fftcplx`, `fftcplxinv`, `fftreal`, `hanncplx`, `hannreal`, `powerspectrum`

---

---

## flash\_erasechip

---

---

```
void flash_erasechip( FlashDescriptor * fd );
```

### DESCRIPTION

Erases an entire flash memory chip.

**Note:** `fd` must have already been initialized with `flash_init` before calling this function. See `flash_init` description for further restrictions.

### PARAMETERS

`fd`                      Pointer to flash descriptor of the chip to erase.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasector`, `flash_gettype`, `flash_init`, `flash_read`,  
`flash_readsector`, `flash_sector2xwindow`, `flash_writesector`

---

---

## flash\_erasesector

---

---

```
int flash_erasesector( FlashDescriptor * fd, word which );
```

### DESCRIPTION

Erases a sector of a flash memory chip.

**Note:** `fd` must have already been initialized with `flash_init` before calling this function. See `flash_init` description for further restrictions.

### PARAMETERS

<code>fd</code>	Pointer to flash descriptor of the chip to erase a sector of.
<code>which</code>	The sector to erase.

### RETURN VALUE

0: Success.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_gettype`, `flash_init`, `flash_read`,  
`flash_readsector`, `flash_sector2xwindow`, `flash_writesector`



---

---

## flash\_gettype

---

---

```
int flash_gettype( FlashDescriptor * fd );
```

### DESCRIPTION

Returns the 16-bit flash memory type of the flash memory.

**Note:** `fd` must have already been initialized with `flash_init` before calling this function. See `flash_init` description for further restrictions.

### PARAMETERS

`fd`                      The FlashDescriptor of the memory to query.

### RETURN VALUE

The integer representing the type of the flash memory.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_erasector`, `flash_init`, `flash_read`,  
`flash_readsector`, `flash_sector2xwindow`, `flash_writesector`

---

---

## flash\_init

---

---

```
int flash_init( FlashDescriptor * fd, int mb3cr );
```

### DESCRIPTION

Initializes an internal data structure of type `FlashDescriptor` with information about the flash memory chip. The Memory Interface Unit bank register (MB3CR) will be assigned the value of `mb3cr` whenever a function accesses the flash memory referenced by `fd`. See the *Rabbit 2000 Users Manual* for the correct chip select and wait state settings.

**Note:** Improper use of this function can cause your program to be overwritten or operate incorrectly. This and the other flash memory access functions should not be used on the same flash memory that your program resides on, nor should they be used on the same region of a second flash memory where a file system resides.

Use `WriteFlash()` to write to the primary flash memory.

### PARAMETERS

<b>fd</b>	This is a pointer to an internal data structure that holds information about a flash memory chip.
<b>mb3cr</b>	This is the value to set MB3CR to whenever the flash memory is accessed. 0xc2 (i.e., CS2, /OE0, /WE0, 0 WS) is a typical setting for the second flash memory on the TCP/IP Dev Kit, the Intellicom, the Advanced Ethernet Core, and the RabbitLink.

### RETURN VALUE

- 0: Success.
- 1: Invalid flash memory type.
- 1: Attempt made to initialize primary flash memory.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_erasector`, `flash_gettype`, `flash_read`,  
`flash_readsector`, `flash_sector2xwindow`, `flash_writesector`

---

---

## flash\_read

---

---

```
int flash_read( FlashDescriptor * fd, word sector, word offset,
               unsigned long buffer, word length );
```

### DESCRIPTION

Reads data from the flash memory and stores it in `buffer`.

**Note:** `fd` must have already been initialized with `flash_init` before calling this function. See the `flash_init` description for further restrictions.

### PARAMETERS

<code>fd</code>	The <code>FlashDescriptor</code> of the flash memory to read from.
<code>sector</code>	The sector of the flash memory to read from.
<code>offset</code>	The displacement, in bytes, from the beginning of the sector to start reading at.
<code>buffer</code>	The physical address of the destination buffer. TIP: A logical address can be changed to a physical with the function <code>paddr</code> .
<code>length</code>	The number of bytes to read.

### RETURN VALUE

0: Success.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_erasector`, `flash_gettype`, `flash_init`,  
`flash_readsector`, `flash_sector2xwindow`, `flash_writesector`, `paddr`

---

---

## flash\_readsector

---

---

```
int flash_readsector( FlashDescriptor * fd, word sector, unsigned
    long buffer );
```

### DESCRIPTION

Reads the contents of an entire sector of flash memory into a buffer.

**Note:** `fd` must have already been initialized with `flash_init` before calling this function. See `flash_init` description for further restrictions.

### PARAMETERS

<b>fd</b>	The <code>FlashDescriptor</code> of the flash memory to read from.
<b>sector</b>	The source sector to read.
<b>buffer</b>	The physical address of the destination buffer. TIP: A logical address can be changed to a physical with the function <code>paddr()</code> .

### RETURN VALUE

0: Success.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_erasector`, `flash_gettype`, `flash_init`,  
`flash_read`, `flash_sector2xwindow`, `flash_writesector`

---

---

## flash\_sector2xwindow

---

---

```
void * flash_sector2xwindow( FlashDescriptor * fd, word sector );
```

### DESCRIPTION

This function sets the MB3CR and XPC value so the requested sector falls within the XPC window. The MB3CR is the Memory Interface Unit bank register. XPC is one of four Memory Management Unit registers. See `flash_init` description for restrictions.

### PARAMETERS

<code>fd</code>	The FlashDescriptor of the flash memory.
<code>sector</code>	The sector to set the XPC window to.

### RETURN VALUE

The logical offset of the sector.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_erasector`, `flash_gettype`, `flash_init`,  
`flash_read`, `flash_readsector`, `flash_writesector`

---

---

## flash\_writesector

---

---

```
int flash_writesector( FlashDescriptor * fd, word sector, unsigned
    long buffer );
```

### DESCRIPTION

Writes the contents of `buffer` to `sector` on the flash memory referenced by `fd`.

**Note:** `fd` must have already been initialized with `flash_init` before calling this function. See `flash_init` description for further restrictions.

### PARAMETERS

<b>fd</b>	The <code>FlashDescriptor</code> of the flash memory to write to.
<b>sector</b>	The destination sector.
<b>buffer</b>	The physical address of the source. TIP: A logical address can be changed to a physical address with the function <code>paddr()</code> .

### RETURN VALUE

0: Success.

### LIBRARY

FLASH.LIB

### SEE ALSO

`flash_erasechip`, `flash_erasector`, `flash_gettype`, `flash_init`,  
`flash_read`, `flash_readsector`, `flash_sector2xwindow`

---

---

## floor

---

---

```
float floor( float x );
```

### DESCRIPTION

Computes the largest integer less than or equal to the given number.

### PARAMETERS

**x**                      Value to round down.

### RETURN VALUE

Rounded down value.

### LIBRARY

MATH.LIB

### SEE ALSO

ceil, fmod

---

---

## fmod

---

---

```
float fmod( float x, float y );
```

### DESCRIPTION

Calculates modulo math.

### PARAMETERS

**x**                      Dividend

**y**                      Divisor

### RETURN VALUE

Returns the remainder of  $x/y$ . The remaining part of  $x$  after all multiples of  $y$  have been removed. For example, if  $x$  is 22.7 and  $y$  is 10.3, the integral division result is 2. Then the remainder is:  $22.7 - 2 \times 10.3 = 2.1$ .

### LIBRARY

MATH.LIB

### SEE ALSO

ceil, floor

---

---

## `fopen_rd (FS1)`

---

---

```
int fopen_rd( File * f, FileNumber fnum );
```

### DESCRIPTION

Opens a file for reading.

### PARAMETERS

<b>f</b>	A pointer to the file to read.
<b>fnum</b>	A number in the range 1 to 127 inclusive that identifies the file in the flash file system.

### RETURN VALUE

0: Success.  
1: Failure.

### LIBRARY

`FILESYSTEM.LIB`



---

---

## `fopen_rd (FS2)`

---

---

```
int fopen_rd( File * f, FileNumber name );
```

### DESCRIPTION

Open file for reading only. See `fopen_wr()` for a more detailed description.

### PARAMETERS

<code>f</code>	Pointer to file descriptor (uninitialized).
<code>name</code>	File number (1 to 255 inclusive).

### RETURN VALUE

0: Success.  
!0: Failure.

### ERRNO VALUES

ENOENT - File does not exist, or metadata extent number does not match an existing file.

### LIBRARY

`fs2.lib`

### SEE ALSO

`fclose`, `fopen_wr (FS2)`

---

---

## `fopen_wr (FS1)`

---

---

```
int fopen_wr( File * f, FileNumber fnum );
```

### DESCRIPTION

Opens a file for writing.

### PARAMETERS

<b>f</b>	A pointer to the file to write.
<b>fnum</b>	A number in the range 1 to 127 inclusive that identifies the file in the flash file system.

### RETURN VALUE

0: Success.  
1: Failure.

### LIBRARY

FILESYSTEM.LIB

---

---

## `fopen_wr (FS2)`

---

---

```
int fopen_wr( File * f, FileNumber name );
```

### DESCRIPTION

Open file for read or write. The given file number is composed of two parts: the low byte contains the file number (1 to 255 inclusive) and the high byte, if not zero, contains the metadata extent number. If the extent number is zero, it defaults to the correct metadata extent - this is for the purpose of validating an expected extent number. Most applications should just pass the file number with zero high byte.

A file may be opened multiple times, with a different file descriptor pointer for each call, which allows the file to be read or written at more than one position at a time. A reference count for the actual file is maintained, so that the file can only be deleted when all file descriptors referring to this file are closed.

`fopen_wr()` or `fopen_rd()` must be called before any other function from this library is called that requires a `File` pointer. The "current position" is set to zero i.e. the start of the file.

When a file is created, it is automatically opened for writing thus a subsequent call to `fopen_wr()` is redundant.

### PARAMETERS

<code>f</code>	Pointer to file descriptor (uninitialized).
<code>name</code>	File number (1 to 255 inclusive).

### RETURN VALUE

0: Success.  
!0: Failure.

### ERRNO VALUES

ENOENT - File does not exist, or metadata extent number does not match an existing file.

### LIBRARY

fs2.lib

### SEE ALSO

`fclose`, `fopen_rd (FS2)`

---

---

## forceSoftReset

---

---

```
void forceSoftReset( void );
```

### DESCRIPTION

Forces the board into a software reset by jumping to the start of the BIOS.

### LIBRARY

SYS.LIB

---

---

## fread (FS1)

---

---

```
int fread( File * f, char * buf, int len );
```

### DESCRIPTION

Reads `len` bytes from a file pointed to by `f`, starting at the current offset into the file, into `buffer`. Data is read into buffer pointed to by `buf`.

### PARAMETERS

<code>f</code>	A pointer to the file to read from.
<code>buf</code>	A pointer to the destination buffer.
<code>len</code>	Number of bytes to copy.

### RETURN VALUE

Number of bytes read.

### LIBRARY

FILESYSTEM.LIB

---

---

## **fread (FS2)**

---

---

```
int fread( File * f, void * buf, int len );
```

### **DESCRIPTION**

Read data from the “current position” of the given file. When the file is opened, the current position is 0, meaning the start of the file. Subsequent reads or writes advance the position by the number of bytes read or written. `fseek()` can also be used to position the read point.

If the application permits, it is much more efficient to read multiple data bytes rather than reading one-by-one.

### **PARAMETERS**

<b>f</b>	Pointer to file descriptor (initialized by <code>fopen_rd()</code> , <code>fopen_wr()</code> or <code>fcreate()</code> ).
<b>buf</b>	Data buffer located in root data memory or stack. This must be dimensioned with at least <code>len</code> bytes.
<b>len</b>	Length of data to read (0 to 32767 inclusive).

### **RETURN VALUE**

`len`: Success.

`<len`: Partial success. Returns amount successfully read. `errno` gives further details (probably 0 meaning that end-of-file was encountered).

0: Failure, or `len` was zero.

### **LIBRARY**

FS2.LIB

### **ERRNO VALUES**

EBADFD - File descriptor not opened.

EINVAL - `len` less than zero.

0 - Success, but `len` was zero or EOF was reached prior to reading `len` bytes.

EIO - I/O error.

### **SEE ALSO**

`fseek (FS2)`, `fwrite (FS2)`

---

---

## frexp

---

---

```
float frexp( float x, int * n );
```

### DESCRIPTION

Splits  $x$  into a fraction and exponent,  $f * ( 2^n )$ .

### PARAMETERS

<b>x</b>	Number to split
<b>n</b>	An integer

### RETURN VALUE

The function returns the exponent in the integer  $*n$  and the fraction between 0.5, inclusive and 1.0.

### LIBRARY

MATH.LIB

### SEE ALSO

`exp`, `ldexp`

---

---

## fs\_format (FS1)

---

---

```
int fs_format( long reserveblocks, int num_blocks, unsigned long
              wearlevel );
```

### DESCRIPTION

Initializes the internal data structures and file system. All blocks in the file system are erased.

### PARAMETERS

<b>reserveblocks</b>	Starting address of the flash file system. When <code>FS_FLASH</code> is defined this value should be 0 or a multiple of the block size. When <code>FS_RAM</code> is defined this parameter is ignored.
<b>num_blocks</b>	The number of blocks to allocate for the file system. With a default block size of 4096 bytes and a 256K flash memory, this value might be 64.
<b>wearlevel</b>	This value should be 1 on a new flash memory, and some higher value on an unformatted used flash memory. If you are reformatting a flash memory you can set <code>wearlevel</code> to 0 to keep the old wear leveling.

### RETURN VALUE

0: Success.  
1: Failure.

### LIBRARY

`FILESYSTEM.LIB`

### EXAMPLE

This program can be found in `samples/filesystem/format.c`.

```
#define FS_FLASH
#include "filesystem.lib"
#define RESERVE 0
#define BLOCKS 64
#define WEAR 1

main() {
    if(fs_format(RESERVE, BLOCKS, WEAR)) {
        printf("error formatting flash\n");
    } else {
        printf("flash successfully formatted\n");
    }
}
```

---

---

## `fs_format (FS2)`

---

---

```
int fs_format( long reserveblocks, int num_blocks, unsigned wearlevel
    );
```

### DESCRIPTION

Format all extents of the file system. This must be called after calling `fs_init()`. Only extents that are not defined as reserved are formatted. All files are deleted.

### PARAMETERS

<code>reserveblocks</code>	Must be zero. Retained for backward compatibility.
<code>num_blocks</code>	Ignored (backward compatibility).
<code>wearlevel</code>	Initial wearlevel value. This should be 1 if you have a new flash, and some larger number if the flash is used. If you are reformatting a flash, you can use 0 to use the old flash wear levels.

### RETURN VALUE

0: Success.  
!0: Failure.

### ERRNO VALUES

EINVAL - the `reserveblocks` parameter was non-zero.  
EBUSY - one or more files were open.  
EIO - I/O error during format. If this occurs, retry the format operation. If it fails again, there is probably a hardware error.

### SEE ALSO

`fs_init (FS2)`, `lx_format`



---

---

## `fs_init (FS1)`

---

---

```
int fs_init( long reserveblocks, int num_blocks );
```

### DESCRIPTION

Initialize the internal data structures for an existing file system. Blocks that are used by a file are preserved and checked for data integrity.

### PARAMETERS

**reserveblocks** Starting address of the flash file system. When `FS_FLASH` is defined this value should be 0 or a multiple of the block size. When `FS_RAM` is defined this parameter is ignored.

**num\_blocks** The number of blocks that the file system contains. By default the block size is 4096 bytes.

### RETURN VALUE

0: Success.  
1: Failure.

### LIBRARY

`FILESYSTEM.LIB`

---

---

## `fs_init (FS2)`

---

---

```
int fs_init( long reserveblocks, int num_blocks );
```

### DESCRIPTION

Initialize the filesystem. The static structure `_fs` contains information that defines the number and parameters associated with each extent or “partition.” This function must be called before any of the other functions in this library, except for `fs_setup()`, `fs_get*_lx()` and `fs_get_lx_size()`.

Pre-main initialization will create up to 3 devices:

- The second flash device (if available on the board)
- Battery-backed SRAM (if `FS2_RAM_RESERVE` defined)
- The first (program) flash (if both `XMEM_RESERVE_SIZE` and `FS2_USE_PROGRAM_FLASH` defined)

The LX numbers of the default devices can be obtained using the `fs_get_flash_lx()`, `fs_get_ram_lx()` and `fs_get_other_lx()` calls. If none of these devices can be set up successfully, `fs_init()` will return `ENOSPC` when called.

This function performs complete consistency checks and, if necessary, fixups for each LX. It may take up to several seconds to run. It should only be called once at application initialization time.

**Note:** When using  $\mu$ C/OS-II, `fs_init()` must be called before `OSInit()`.

### PARAMETERS

`reserveblocks`    Must be zero. Retained for backward compatibility.

`num_blocks`      Ignored (backward compatibility).

### RETURN VALUE

0: Success.

!0: Failure.

### ERRNO VALUES

`EINVAL` - the `reserveblocks` parameter was non-zero.

`EIO` - I/O error. This indicates a hardware problem.

`ENOMEM` - Insufficient memory for required buffers.

`ENOSPC` - No valid extents obtained e.g. there is no recognized flash or RAM memory device available.

### LIBRARY

`fs2.lib`

### SEE ALSO

`fs_setup (FS2)`, `fs_get_flash_lx (FS2)`

---

---

## **fs\_reserve\_blocks (FS1)**

---

---

```
int fs_reserve_blocks( int blocks );
```

### **DESCRIPTION**

Sets up a number of blocks that are guaranteed to be available for privileged files. A privileged file has an identifying number in the range 128 through 143. This function is not needed in most cases. If it is used, it should be called immediately after `fs_init` or `fs_format`.

### **PARAMETERS**

**blocks**            Number of blocks to reserve.

### **RETURN VALUE**

0: Success.  
1: Failure.

### **LIBRARY**

FILESYSTEM.LIB

---

---

## **fsck (FS1)**

---

---

```
int fsck( int flash );
```

### **DESCRIPTION**

Check the filesystem for errors

### **PARAMETERS**

**flash**            A bitmask indicating which checks to NOT perform. The following checks are available:

FSCK\_HEADERS - Block headers.

FSCK\_CHECKSUMS - Data checksums.

FSCK\_VERSION - Block versions, from a failed write.

### **RETURN VALUE**

0: Success.  
!0: Failure, this is a bitmask indicating which checks failed.

### **LIBRARY**

FILESYSTEM.LIB

---

---

## **fseek (FS1)**

---

---

```
int fseek( File * f, long to, char whence );
```

### **DESCRIPTION**

Places the read pointer at a desired location in the file.

### **PARAMETERS**

<b>f</b>	A pointer to the file to seek into.
<b>to</b>	The number of bytes to move the read pointer. This can be a positive or negative number.
<b>whence</b>	The location in the file to offset from. This is one of the following constants.  SEEK_SET - Seek from the beginning of the file. SEEK_CUR - Seek from the current read position in the file. SEEK_END - Seek from the end of the file.

### **EXAMPLE**

To seek to 10 bytes from the end of the file `f`, use

```
fseek(f, -10, SEEK_END);
```

To rewind the file `f` by 5 bytes, use

```
fseek(f, -5, SEEK_CUR);
```

### **RETURN VALUE**

0: Success.  
1: Failure.

### **LIBRARY**

FILESYSTEM.LIB

---

---

## **fseek (FS2)**

---

---

```
int fseek( File * f, long where, char whence );
```

### **DESCRIPTION**

Set the current read/write position of the file. Bytes in a file are sequentially numbered starting at zero. If the current position is zero, then the first byte of the file will be read or written. If the position equals the file length, then no data can be read, but any write will append data to the file.

`fseek()` allows the position to be set relative to the start or end of the file, or relative to its current position.

In the special case of `SEEK_RAW`, an unspecified number of bytes beyond the known end-of-file may be readable. The actual amount depends on the amount of space left in the last internal block of the file. This mode only applies to reading, and is provided for the purpose of data recovery in the case that the application knows more about the file structure than the filesystem.

### **PARAMETERS**

<b>f</b>	Pointer to file descriptor (initialized by <code>fopen_rd()</code> , <code>fopen_wr()</code> or <code>fcreate()</code> ).
<b>where</b>	New position, or offset.
<b>whence</b>	One of the following values: SEEK_SET: 'where' (non-negative only) is relative to start of file. SEEK_CUR: 'where' (positive or negative) is relative to the current position. SEEK_END: 'where' (non-positive only) is relative to the end of the file. SEEK_RAW: Similar to <code>SEEK_END</code> , except the file descriptor is set in a special mode which allows reading beyond the end of the file.

### **RETURN VALUE**

0: Success.

!0: The computed position was outside of the current file contents, and has been adjusted to the nearest valid position.

### **ERRNO VALUES**

None.

### **LIBRARY**

FS2.LIB

### **SEE ALSO**

`ftell (FS2)`, `fread (FS2)`, `fwrite (FS2)`

---

---

## `fs_get_flash_lx (FS2)`

---

---

```
FSLXnum fs_get_flash_lx( void );
```

### DESCRIPTION

Returns the logical extent number of the preferred flash device. This is the second flash if one is available on your hardware, otherwise it is the reserved area in your program flash. In order for the program flash to be available for use by the file system, you must define two constants: the first constant is `XMEM_RESERVE_SIZE` near the top of `BIOS\RABBITBIOS.C`. This value is set to the amount of program flash to reserve (in bytes). This is required by the BIOS. The second constant is set in your code before `#use "fs2.lib"`. `FS2_USE_PROGRAM_FLASH` must be defined to the number of KB (1024 bytes) that will actually be used by the file system. If this is set to a larger value than the actual amount of reserved space, then only the actual amount will be used.

The sample program `SAMPLES\FILESYSTEM\FS2INFO.C` demonstrates use of this function.

This function may be called before calling `fs_init()`.

### RETURN VALUE

- 0: There is no flash file system available.
- !0: Logical extent number of the preferred flash.

### LIBRARY

`FS2.lib`

### SEE ALSO

`fs_get_ram_lx (FS2)`, `fs_get_other_lx (FS2)`

---

---

## `fs_get_lx (FS2)`

---

---

```
FSLXnum fs_get_lx( int meta );
```

### DESCRIPTION

Return the current extent (LX) number for file creation. Each file has two parts: the main bulk of data, and the metadata which is a relatively small, fixed, amount of data used to journal changes to the file. Both data and metadata can reside on the same extent, or they may be separated.

### PARAMETERS

<code>meta</code>	1: return logical extent number for metadata. 0: return logical extent number for data.
-------------------	--

### RETURN VALUE

Logical extent number.

### LIBRARY

`FS2.lib`

### SEE ALSO

`fcreate (FS2)`, `fs_set_lx (FS2)`

---

---

## fs\_get\_lx\_size (FS2)

---

---

```
long fs_get_lx_size( FSLXnum lxn, int all, word ls_shift );
```

### DESCRIPTION

Returns the size of the specified logical extent, in bytes. This information is useful when initially partitioning an LX, or when estimating the capacity of an LX for user data. `all` is a flag which indicates whether to return the total data capacity (as if all current files were deleted) or whether to return just the available data capacity. The return value accounts for the packing efficiency which will be less than 100% because of the bookkeeping overhead. It does not account for the free space required when any updates are performed; however this free space may be shared by all files on the LX. It also does not account for the space required for file metadata. You can account for this by adding one logical sector for each file to be created on this LX. You can also specify that the metadata be stored on a different LX by use of `fs_set_lx()`.

This function may be called either before or after `fs_init()`. If called before, then the `ls_shift` parameter must be set to the value to be used in `fs_setup()`, since the LS size is not known at this point. `ls_shift` can also be passed as zero, in which case the default size will be assumed. `all` must be non-zero if called before `fs_init()`, since the number of files in use is not yet known.

### PARAMETERS

<b>lxn</b>	Logical extent number to query.
<b>all</b>	Boolean: 0 for current free capacity only, 1 for total. Must use 1 if calling before <code>fs_init()</code> .
<b>ls_shift</b>	Logical sector shift i.e. log base 2 of LS size (6 to 13); may be zero to use default.

### RETURN VALUE

- 0: The specified LX does not exist.
- !0: Capacity of the LX in bytes.

### LIBRARY

FS2.lib



---

---

## `fs_get_other_lx (FS2)`

---

---

```
FSLXnum fs_get_other_lx( void );
```

### DESCRIPTION

Returns the logical extent number of the non-preferred flash device. If it exists, this is usually the program flash. See the description under `fs_get_flash_lx()` for details about setting up the program flash for use by the filesystem.

The sample program `Samples\FILESYSTEM\F2INFO.C` demonstrates use of this function.

This function may be called before calling `fs_init()`.

### RETURN VALUE

0: There is no other flash filesystem available.

!0: Logical extent number of the non-preferred flash.

### LIBRARY

`FS2.LIB`

### SEE ALSO

`fs_get_ram_lx (FS2)`, `fs_get_flash_lx (FS2)`

---

---

## `fs_get_ram_lx (FS2)`

---

---

```
FSLXnum fs_get_ram_lx( void );
```

### DESCRIPTION

Return the logical extent number of the RAM file system device. This is only available if you have defined `FS2_RAM_RESERVE` to a non-zero number of bytes in the BIOS.

A RAM filesystem is only really useful if you have battery-backed SRAM on the board. You can still use a RAM file system on volatile RAM, but of course files will not persist over power cycles and you should explicitly format the RAM filesystem at power-up.

The sample program `Samples\FILESYSTEM\FS2INFO.C` demonstrates use of this function.

This function may be called before calling `fs_init()`.

### RETURN VALUE

0: There is no RAM filesystem available.

!0: Logical extent number of the RAM device.

### LIBRARY

`FS2.LIB`

### SEE ALSO

`fs_get_flash_lx (FS2)`, `fs_get_other_lx (FS2)`

---

---

## `fs_set_lx (FS2)`

---

---

```
int fs_set_lx( FSLXnum meta, FSLXnum data );
```

### DESCRIPTION

Sets the default logical extent (LX) numbers for file creation. Each file has two parts: the main bulk of data, and the metadata which is a relatively small, fixed amount of data used to journal changes to the file. Both data and metadata can reside on the same extent, or they may be separated. The metadata, no matter where it is located, consumes one sector.

The file creation functions allow the metadata extent to be explicitly specified (in the high byte of the file number), however it is usually easier to call `fs_set_lx()` to set appropriate defaults. Calling `fs_set_lx()` is the only way to specify the data extent.

If `fs_set_lx()` is never called, both data and metadata will default to the first non-reserved extent number.

### PARAMETERS

<code>meta</code>	Extent number for metadata.
<code>data</code>	Extent number for data.

### RETURN VALUE

0: Success.  
!0: Error, e.g. non-existent LX number.

### ERRNO VALUES

ENODEV - no such extent number, or extent is reserved.

### LIBRARY

FS2.LIB

### SEE ALSO

`fcreate (FS2)`

---

---

## fs\_setup (FS2)

---

---

```
FSLXnum fs_setup( FSLXnum lxn, word ls_shift, int reserve_it, void *
    rfu, int partition_it, word part, word part_ls_shift, int
    part_reserve, void * part_rfu );
```

### DESCRIPTION

To modify or add to the default extents, this function must be called before calling `fs_init()`. If called after `fs_init()`, the filesystem will be corrupted.

`fs_setup()` runs in one of two basic modes, determined by the `partition_it` parameter. If `partition_it` is non-zero, then the specified extent (`lxn`, which must exist), is split into two extents according to the given proportions. If `partition_it` is zero, then the specified extent must not exist; it is created. This use is beyond the scope of this note, since it involves filesystem internals. The partitioning usage is described here.

`partition_it` may be `FS_MODIFY_EXTENT` in which case the base extent, `lxn`, is modified to use the specified `ls_shift` and `reserve_it` parameters (the other parameters are ignored).

`partition_it` may be set to `FS_PARTITION_FRACTION` (other values reserved). This causes extent number `lxn` to be split. The first half is still referred to as extent `lxn`, and the other half is assigned a new extent number, which is returned.

The base extent number may itself have been previously partitioned, or it should be 1 for the 2nd flash device, or possibly 2 for the NVRAM device.

### PARAMETERS

<b>lxn</b>	Base extent number to partition or modify.
<b>ls_shift</b>	New logical sector size to assign to base partition, or zero to not alter it. This is expressed as the log base 2 of the desired size, and must be a number between 6 and 13 inclusive.
<b>reserve_it</b>	TRUE if base partition is to be marked reserved.
<b>rfu</b>	A pointer reserved for future use. Pass as null.
<b>partition_it</b>	Must be set to <code>FS_PARTITION_FRACTION</code> or <code>FS_MODIFY_EXTENT</code> . The following parameters are ignored if this parameter is not <code>FS_PARTITION_FRACTION</code> .

---

---

## **fs\_setup (FS2) (cont'd)**

---

---

<b>part</b>	The fraction of the existing base extent to assign to the new extent. This number is expressed as a fixed-point binary number with the binary point to the left of the MSB e.g. 0x3000 assigns 3/16 of the base extent to the new partition, updating the base extent to 13/16 of its original size. The nearest whole number of physical sectors is used for each extent.
<b>part_ls_shift</b>	Logical sector size to assign to the new extent, or zero to use the same LS size as the base extent. Expressed in same units as parameter 2.
<b>part_reserve</b>	TRUE if the new extent is to be reserved.
<b>part_rfu</b>	A pointer reserved for future use. Pass as null.

### **RETURN VALUE**

- 0: Failure, extent could not be partitioned.
- !0: Success, number of the new extent, or same as lxd for existing extent modification.

### **ERRNO VALUES**

- ENOSPC - one or other half would contain an unusably small number of logical sectors, or the extent table is full. In the latter case, `#define FS_MAX_LX` to a larger value.
- EINVAL - `partition_it` set to an invalid value, or other parameter invalid.
- ENODEV - specified base extent number not defined.

### **LIBRARY**

FS2.LIB

### **SEE ALSO**

`fs_init (FS2)`

---

---

## **fs\_sync (FS2)**

---

---

```
int fs_sync( void );
```

### **DESCRIPTION**

Flush any buffers retained in RAM to the underlying hardware device. The file system does not currently perform any buffering, however future revisions of this library may introduce buffering to improve performance. This function is similar to `fflush()`, except that the entire file system is synchronized instead of the data for just one file. Use `fs_sync()` in preference to `fflush()` if there is only one extent in the filesystem.

### **RETURN VALUE**

0: Success.  
!0: Failure.

### **ERRNO VALUES**

EIO - I/O error.

### **LIBRARY**

FS2.LIB

### **SEE ALSO**

`fflush (FS2)`

---

---

## `ftell (FS1)`

---

---

```
long ftell( File * f );
```

### DESCRIPTION

Gets the offset from the beginning of a file that the read pointer is currently at.

TIP: `ftell()` can be used with `fseek()` to find the length of a file.

```
fseek(f, 0, SEEK_END); // seek to the end of the file
FileLength = ftell(f); // find the length of the file
```

### PARAMETERS

**f**                      A pointer to the file to query.

### RETURN VALUE

The offset in bytes of the read pointer from the beginning of the file: Success.  
-1: Failure.

### LIBRARY

FILESYSTEM.LIB

---

---

## `ftell (FS2)`

---

---

```
long ftell( File * f );
```

### DESCRIPTION

Return the current read/write position of the file. Bytes in a file are sequentially numbered starting at zero. If the current position is zero, then the first byte of the file will be read or written. If the position equals the file length, then no data can be read, but any write will append data to the file.

Note that no checking is done to see if the file descriptor is valid. If the File is not actually open, the return value will be random.

### PARAMETERS

**f**                    Pointer to file descriptor (initialized by `fopen_rd()`, `fopen_wr()` or `fcreate()`).

### RETURN VALUE

Current read/write position (0 to length-of-file).

### ERRNO VALUES

None

### LIBRARY

`fs2.lib`

### SEE ALSO

`fseek (FS2)`



---

---

## fshift

---

---

```
int fshift( File * f, int len, void * buf );
```

### DESCRIPTION

Delete data from the start of a file opened for writing. Optionally, the data that was removed can be read into a buffer. The “current position” of the file descriptor is adjusted to take account of the changed file offsets. If the current position is pointing into the data that is removed, then it is set to zero, i.e., the start of data immediately after the deleted section.

The specified file must not be opened with other file descriptors, otherwise an EBUSY error is returned. The exception to this is if `FS2_SHIFT_DOESNT_UPDATE_FPOS` is defined before `#use fs2.lib`. If defined, multiple file descriptors can be opened, but their current position will not be updated if `fshift()` is used. In this case, the application should explicitly use `fseek()` on all file descriptors open on this file (including the one used to perform the `fshift()`). If this is not done, then their current position is effectively advanced by the number of characters shifted out by the `fshift()`.

The purpose of this function is to make it easy to implement files which worm their way through the filesystem: adding at the head and removing at the tail, such that the total file size remains approximately constant.

Surprisingly, it is possible for an out-of-space error to occur, since the addition of the journaling (meta-data) entry for the shift operation may cause an error before deleted blocks (if any) are made available.

### PARAMETERS

<b>f</b>	Pointer to file descriptor (initialized by <code>fopen_wr()</code> or <code>fcreate()</code> ).
<b>len</b>	Length of data to remove (0 to 32767 inclusive).
<b>*buf</b>	Data buffer located in root data memory or stack. This must be dimensioned with at least <code>len</code> bytes. This parameter may also be null if the deleted data is not needed.

---

---

## `fshift` (cont'd)

---

---

### RETURN VALUE

`len`: Success.  
<`len`: Partial success - returns amount successfully deleted. `errno` gives further details (probably `ENOSPC`)  
0: Error or `len` was zero.

### ERRNO VALUES

`EBADFD` - File descriptor not opened, or is read-only.  
`EINVAL` - `len` less than zero.  
0 - Success, but `len` was zero.  
`EIO` - I/O error.  
`ENOSPC` - extent out of space.  
`EBUSY` - file opened more than once. This is only possible if  
`FS2_SHIFT_DOESNT_UPDATE_FPOS` is not defined, which is the default case.

### LIBRARY

`FS2.LIB`

### SEE ALSO

`fread` (`FS2`), `fwrite` (`FS2`)

---

---

## **fwrite (FS1)**

---

---

```
int fwrite( File * f, char * buf, int len );
```

### **DESCRIPTION**

Appends `len` bytes from the source buffer to the end of the file.

### **PARAMETERS**

<b>f</b>	A pointer to the file to write to.
<b>buf</b>	A pointer to the source buffer.
<b>len</b>	The number of bytes to write.

### **RETURN VALUE**

The number of bytes written: Success.  
0: Failure.

### **LIBRARY**

FILESYSTEM.LIB

---

---

## fwrite (FS2)

---

---

```
int fwrite( File * f, void * buf, int len );
```

### DESCRIPTION

Write data to file opened for writing. The data is written starting at the current position. This is zero (start of file) when it is opened or created, but may be changed by `fread()`, `fwrite()`, `fshift()` or `fseek()` functions. After writing the data, the current position is advanced to the position just after the last byte written. Thus, sequential calls to `fwrite()` will add or append data contiguously.

Unlike the previous file system (`FILESYSTEM.LIB`), this library allows files to be overwritten not just appended. Internally, overwrite and append are different operations with differing performance, depending on the underlying hardware. Generally, appending is more efficient especially with byte-writable flash memory. If the application allows, it is preferable to use append/shift rather than overwrite. In order to ensure that data is appended, use `fseek(f, 0, SEEK_END)` before calling `fwrite()`.

The same current-position pointer is used for both read and write. If interspersing read and write, then `fseek()` should be used to ensure the correct position for each operation. Alternatively, the same file can be opened twice, with one descriptor used for read and the other for write. This precludes use of `fshift()`, since it does not tolerate shared files.

### PARAMETERS

<b>f</b>	Pointer to file descriptor (initialized by <code>fopen_wr()</code> or <code>fcreate()</code> ).
<b>buf</b>	Data buffer located in root data memory or stack.
<b>len</b>	Length of data (0 to 32767 inclusive).

### RETURN VALUE

len: Success.  
<len: Partial success. Returns amount successfully written. `errno` gives details.  
0: Failure, or len was zero.

### ERRNO VALUES

EBADFD - File descriptor not opened, or is read-only.  
EINVAL - len less than zero.  
0 - Success, but len was zero.  
EIO - I/O error.  
ENOSPC - extent out of space.

### LIBRARY

fs2.LIB

### SEE ALSO

`fread (FS2)`

---

---

## ftoa

---

---

```
int ftoa( float f, char * buf );
```

### DESCRIPTION

Converts a float number to a character string.

The character string only displays the mantissa up to 9 digits, no decimal points, and a minus sign if `f` is negative. The function returns the exponent (of 10) that should be used to compensate for the string: `ftoa(1.0, buf)` yields `buf="100000000"` and returns `-8`.

### PARAMETERS

<code>f</code>	Float number to convert.
<code>buf</code>	Converted string. The string is no longer than 10 characters long.

### RETURN VALUE

The exponent of the number.

### LIBRARY

STDIO.LIB

### SEE ALSO

`utoa`, `itoa`

---

---

## getchar

---

---

```
char getchar( void );
```

### DESCRIPTION

Busy waits for a character to be typed from the stdio window in Dynamic C. The user should make sure only one process calls this function at a time.

### RETURN VALUE

A character typed in the Stdio window in Dynamic C.

### LIBRARY

STDIO.LIB

### SEE ALSO

`gets`, `putchar`

---

---

## getcrc

---

---

```
int getcrc( char * dataarray, char count, int accum );
```

### DESCRIPTION

Computes the Cyclic Redundancy Check (CRC), or check sum, for `count` bytes (maximum 255) of data in buffer. Calls to `getcrc` can be “concatenated” using `accum` to compute the CRC for a large buffer.

### PARAMETERS

<code>dataarray</code>	Data buffer
<code>count</code>	Number of bytes. Maximum is 255.
<code>accum</code>	Base CRC for the data array.

### RETURN VALUE

CRC value.

### LIBRARY

MATH.LIB

---

---

## getdivider19200

---

---

```
char getdivider19200( void );
```

### DESCRIPTION

This function returns a value that is used in baud rate calculations.

The correct value is returned regardless of the compile mode. In separate I&D space mode, the divider value is stored as a define byte in code space, so directly accessing the variable will result in an incorrect load (from constant data space). This function uses the `ldp` instruction, which circumvents the separate I&D default loading scheme so that the correct value is returned.

### RETURN VALUE

The value used in baud rate calculation.

### LIBRARY

SYS.LIB

---

---

## gets

---

---

```
char * gets( char * s );
```

### DESCRIPTION

Waits for a string terminated by <CR> at the stdio window. The string returned is null terminated without the return. The user should make sure only one process calls this function at a time.

### PARAMETERS

<b>s</b>	The input string is put to the location pointed to by the argument <i>s</i> . The caller is responsible to make sure the location pointed to by <i>s</i> is big enough for the string.
----------	--

### RETURN VALUE

Same pointer passed in, but string is changed to be null terminated.

### LIBRARY

STDIO.LIB

### SEE ALSO

puts, getchar

---

---

## **`_GetSysMacroIndex`**

---

---

```
int _GetSysMacroIndex( int n, char * buf, uint32 * value );
```

### **DESCRIPTION**

Skips to the `n`th macro entry and retrieves the macro name (as defined by the compiler), and the value of the macro as defined in the system macro table. The system macro table contains board specific configuration parameters that are defined by the compiler and can be retrieved at runtime through this interface. The flash driver must be initialized and the System ID block must be read before this function will return accurate results.

This function only applies to boards with Version 5 or later System ID blocks.

### **PARAMETERS**

<b><code>n</code></b>	The index in the system macro table.
<b><code>buf</code></b>	Character array to contain and return macro name (copied from system macro table). <b>MUST BE AT LEAST <code>SYS_MACRO_LENGTH</code> bytes</b> or function will overflow buffer and can crash system!
<b><code>value</code></b>	Pointer to macro value to return to caller.

### **RETURN VALUE**

- 0: if successful
- 1: invalid address or range (use to find end of table)
- 2: ID block or macro table invalid

### **LIBRARY**

`IDBLOCK.LIB`

### **SEE ALSO**

`_GetSysMacroValue`



---

---

## **\_GetSysMacroValue**

---

---

```
int _GetSysMacroValue( char * name, long * value );
```

### **DESCRIPTION**

Finds the system table macro named by the first parameter (as defined by the compiler) and retrieves the value of the macro as defined in the system macro table. The system macro table contains board specific configuration parameters that are define by the compiler and can be retrieved at runtime through this interface. The flash driver must be initialized and the System ID block must be read before this function will return accurate results.

See `writeUserBlockArray` for more details.

This function only applies to boards with Version 5 or later System ID blocks.

### **PARAMETERS**

<b>name</b>	Name of System ID block macro (acts as lookup key).
<b>value</b>	Pointer to macro value to return to caller.

### **RETURN VALUE**

- 0: if successful
- 1: Macro name not found
- 2: No valid ID block found (block version 3 or later)
- 3: First parameter is a bad macro name

### **LIBRARY**

IDBLOCK.LIB

### **SEE ALSO**

`writeUserBlockArray`

---

---

## GetVectExtern2000

---

---

```
unsigned GetVectExtern2000( void );
```

### DESCRIPTION

Reads the address of external interrupt table entry. This function really just returns what is present in the table. The return value is meaningless if the address of the external interrupt has not been written.

This function should be used for Rabbit 2000 processors that are marked IQ2T in the 3rd line of text across the face of the chip. It will work for other versions of the Rabbit 2000 but should be deprecated in favor of `GetVectExtern3000()` which allows the use of two external interrupts. (Please see document TN301, “Rabbit 2000 Microprocessor Interrupt Issue,” on the [Rabbit Semiconductor website](#) for more information.)

### RETURN VALUE

Jump address in vector table.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`GetVectIntern`, `SetVectExtern2000`, `SetVectIntern`, `GetVectExtern3000`

---

---

## GetVectExtern3000

---

---

```
unsigned GetVectExtern3000( int interruptNum );
```

### DESCRIPTION

Reads the address of an external interrupt table entry. This function may be used with all Rabbit 3000 processors and all Rabbit 2000 processors with the exception of the ones marked IQ2T in the 3rd line of text across the face of the chip. For those, use the function `GetVectExtern2000()` instead.

`GetVectExtern3000()` returns the value at address:

$$(\text{external vector table base}) + (\text{interruptNum} * 8) + 1$$

### PARAMETER

**interruptNum** Interrupt number. Should be 0 or 1.

### RETURN VALUE

Jump address in vector table.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`SetVectExtern3000`, `SetVectIntern`, `GetVectIntern`, `GetVectExtern2000`

---

---

## GetVectIntern

---

---

```
unsigned GetVectIntern( int vectNum );
```

### DESCRIPTION

Reads the address of the internal interrupt table entry and returns whatever value is at the address:

```
(internal vector table base) + (vectNum*16) + 1
```

### PARAMETER

**vectNum**            Interrupt number; should be 0–15.

### RETURN VALUE

Jump address in vector table.

### LIBRARY

SYS.LIB

### SEE ALSO

GetVectExtern2000, SetVectExtern2000, SetVectIntern

---

---

## gps\_get\_position

---

---

```
int gps_get_position( GPSPosition * newpos, char * sentence );
```

### DESCRIPTION

Parses a sentence to extract position data. This function is able to parse any of the following GPS sentence formats: GGA, GLL or RMC.

### PARAMETERS

**newpos**            A GPSPosition structure to fill.

**sentence**         A string containing a line of GPS data in NMEA-0183 format.

### RETURN VALUE

0: Success.  
-1: Parsing error.  
-2: Sentence marked invalid.

### LIBRARY

gps.lib

---

---

## gps\_get\_utc

---

---

```
int gps_get_utc( struct tm * newtime, char * sentence );
```

### DESCRIPTION

Parses an RMC sentence to extract time data.

### PARAMETERS

<b>newtime</b>	tm structure to fill with new UTC time.
<b>sentence</b>	A string containing a line of GPS data in NMEA-0183 format (RMC sentence).

### RETURN VALUE

0: Success.  
-1: Parsing error.  
-2: Sentence marked invalid.

### LIBRARY

GPS.LIB

---

---

## gps\_ground\_distance

---

---

```
float gps_ground_distance( GPSPosition * a, GPSPosition * b );
```

### DESCRIPTION

Calculates ground distance (in km) between two geographical points. (Uses spherical earth model.)

### PARAMETERS

<b>a</b>	First point.
<b>b</b>	Second point.

### RETURN VALUE

Distance in kilometers.

### LIBRARY

GPS.LIB

---

---

## hanncplx

---

---

```
void hanncplx( int * x, int N, int * blockexp );
```

### DESCRIPTION

Convolve an N-point complex spectrum with the three-point Hann kernel. The filtered spectrum replaces the original spectrum.

The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero-crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

N must be a power of 2 and between 4 and 1024. An invalid N causes a RANGE exception.

### PARAMETERS

<b>x</b>	Pointer to N-element array of complex fractions.
<b>N</b>	Number of complex elements in array x.
<b>blockexp</b>	Pointer to integer block exponent.

### LIBRARY

FFT.LIB

### SEE ALSO

fftcplx, fftcplxinv, fftreal, fftrealinv, hanncplx, powerspectrum

---

---

## hannreal

---

---

```
void hannreal( int * x, int N, int * blockexp );
```

### DESCRIPTION

Convolve an N-point positive-frequency complex spectrum with the three-point Hann kernel. The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero-crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

The imaginary part of the dc term (stored in  $x[1]$ ) is considered to be the real part of the  $f_{max}$  term. The dc and  $f_{max}$  spectral components take part in the convolution along with the other spectral components. The real part of  $f_{max}$  component affects the real part of the  $X[N-1]$  component (and vice versa), and should not arbitrarily be set to zero unless these components are unimportant.

### PARAMETERS

<b>x</b>	Pointer to N-element array of complex fractions.
<b>N</b>	Number of complex elements in array x.
<b>blockexp</b>	Pointer to integer block exponent.

### RETURN VALUE

None. The filtered spectrum replaces the original spectrum.

### LIBRARY

FFT.LIB

### SEE ALSO

`fftcplx`, `fftcplxinv`, `fftreal`, `fftrealinv`, `hanncplx`, `powerspectrum`

---

---

## HDLCabortX

---

---

```
void HDLCabortX( void ); /* Where X is E or F */
```

### DESCRIPTION

Immediately stops any transmission. An HDLC abort code will be sent if the driver was in the middle of sending a packet.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### LIBRARY

```
HDLC_PACKET.LIB
```

---

---

## HDLCcloseX

---

---

```
void HDLCcloseX( void ); /* Where X is E or F */
```

### DESCRIPTION

Disables the HDLC port (E or F). If it was used, the TAT1R resource (timer A1 cascade) is released. This function is non-reentrant.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### LIBRARY

```
HDLC_PACKET.LIB
```

### SEE ALSO

```
TAT1R_SetValue
```



---

---

## HDLCdropX

---

---

```
int HDLCdropX( void ); /* Where X is E or F */
```

### DESCRIPTION

Drops the next received packet, freeing up its buffer. This must be used if the packet has been examined with HDLCpeekX() and is no longer needed. A call to HDLCreceiveX() is the only other way to free up the buffer.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### RETURN VALUE

- 1: Packet dropped.
- 0: No received packets were available.

### LIBRARY

HDLC\_PACKET.LIB

---

---

## HDLCerrorX

---

---

```
int HDLCerrorX( unsigned long * bufptr, int * lenptr );  
/* Where X is E or F */
```

### DESCRIPTION

This function returns a set of possible error flags as an integer. A received packet with errors is automatically dropped.

Masks are used to check which errors have occurred. The masks are:

- HDLC\_NOBUFFER - driver ran out of buffers for received packets.
- HDLC\_OVERRUN - a byte was overwritten and lost before the ISR could retrieve it.
- HDLC\_OVERFLOW - a received packet was too long for the buffers.
- HDLC\_ABORTED - a received packet was aborted by the sender during transmission.
- HDLC\_BADCRC - a packet with an incorrect CRC was received.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### RETURN VALUE

Error flags (see above).

### LIBRARY

HDLC\_PACKET.LIB

---

---

## HDLCExtClockX

---

---

```
void HDLCExtClockE( int ext_clock ) /* Where X is E or F */
```

### DESCRIPTION

Configures HDLC to be either internally (default) or externally clocked. This should be called after HDLCopenX().

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### PARAMETER

<b>ext_clock</b>	1 for externally clocked
	0 for internally clocked

### LIBRARY

HDLC\_PACKET.LIB

---

---

## HDLCopenX

---

---

```
int HDLCopenX( long baud, char encoding, unsigned long buffers, int
  buffer_count, int buffer_size ); /* Where X is E or F */
```

### DESCRIPTION

Opens serial port E or F in HDLC mode. Sets up buffers to hold received packets. This function is intended for use with the Rabbit 3000 and Rabbit 4000. Please see the chip manuals for more details on HDLC and the bit encoding modes to use.

### PARAMETERS

<b>baud</b>	The baud rate for the serial port. Due to imitations in the baud generator, non-standard baud rates will be approximated within 5% of the value requested.
<b>encoding</b>	The bit encoding mode to use. Macro labels for the available options are: <ul style="list-style-type: none"><li>• HDLC_NRZ</li><li>• HDLC_NRZI</li><li>• HDLC_MANCHESTER</li><li>• HDLC_BIPHASE_SPACE</li><li>• HDLC_BIPHASE_MARK</li></ul>
<b>buffers</b>	A pointer to the start of the extended memory block containing the receive buffers. This block must be allocated beforehand by the user. The size of the block should be: $(\# \text{ of buffers}) * ((\text{size of buffer}) + 4)$
<b>buffer_count</b>	The number of buffers in the block pointed to by <code>buffer</code> .
<b>buffer_size</b>	The capacity of each buffer in the block pointed to by <code>buffer</code> .

### RETURN VALUE

1: Actual baud rate is within 5% of the requested baud rate,  
0: Otherwise.

### LIBRARY

HDLC\_PACKET.LIB

### SEE ALSO

SetSerialTATxRValues, TAT1R\_SetValue

---

---

## HDLCpeekX

---

---

```
int HDLCpeekX( unsigned long * bufptr, int * lenptr );  
/* Where X is E or F */
```

### DESCRIPTION

Reports the location and size of the next available received packet if one is available. This function can be used to efficiently inspect a received packet without actually copying it into a root memory buffer. Once inspected, the buffer can be received normally (see HDLCreceiveX ( ) ), or dropped (see HDLCdropX ( ) ).

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

<b>bufptr</b>	Pointer to location in xmem of the received packet.
<b>lenptr</b>	Pointer to the size of the received packet.

### RETURN VALUE

1: The pointers `bufptr` and `lenptr` have been set for the received packet.  
0: No received packets available.

### LIBRARY

HDLC\_PACKET.LIB

---

---

## HDLCreceiveX

---

---

```
int HDLCreceiveX(char *rx_buffer, int length); /* Where X is E or F */
```

### DESCRIPTION

Copies a received packet into `rx_buffer` if there is one. Packets are received in the order they arrive, even if multiple packets are currently stored in buffers.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

**rx\_buffer**      Pointer to the buffer to copy a received packet into.

**length**         Size of the buffer pointed to by `rx_buffer`.

### RETURN VALUE

≥0: Size of received packet.

-1: No packets are available to receive.

-2: The buffer is not large enough for the received packet. In this case, the packet remains in the receive buffer)

### LIBRARY

HDLC\_PACKET.LIB

---

---

## HDLCsendX

---

---

```
int HDLCsendX( char * tx_buffer, int length ); /* Where X is E or F */
```

### DESCRIPTION

Transmits a packet out serial port E or F in HDLC mode. The tx\_buffer is read directly while transmitting, therefore it cannot be altered until a subsequent call to HDLCsendX() returns false, indicating that the driver is done with it.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

**tx\_buffer**        A pointer to the packet to be sent. This buffer must not change while transmitting (see above.)

**length**            The size of the buffer (in bytes).

### RETURN VALUE

1: Sending packet.

0: Cannot send, another packet is currently being transmitted.

### LIBRARY

HDLC\_PACKET.LIB

---

---

## HDLCsendingX

---

---

```
int HDLCsendingX( void ); /* Where X is E or F */
```

### DESCRIPTION

Returns true if a packet is currently being transmitted. This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### RETURN VALUE

1: Currently sending a packet.  
0: Transmitter is idle.

### LIBRARY

HDLC\_PACKET.LIB

---

---

## hexstrtobyte

---

---

```
int hexstrtobyte (char far *p);
```

### DESCRIPTION

Converts two hex characters (0-9A-Fa-f) to a byte.

### RETURN VALUE

The byte (0-255) represented by the two hex characters or -1 on error (invalid character, string less than 2 bytes).

### EXAMPLES

hexstrtobyte("FF") returns 255  
hexstrtobyte("0") returns -1 (error because < 2 characters)  
hexstrtobyte("ABCDEF") returns 0xAB (ignores additional chars)

---

---

## hitwd

---

---

```
void hitwd( void );
```

### DESCRIPTION

Hits the watchdog timer, postponing a hardware reset for 2 seconds. Unless the watchdog timer is disabled, a program must call this function periodically, or the controller will automatically reset itself. If the virtual driver is enabled (which it is by default), it will call `hitwd` in the background. The virtual driver also makes additional “virtual” watchdog timers available.

### LIBRARY

VDRIVER.LIB

---

---

## htoa

---

---

```
char * htoa( int value, char * buf );
```

### DESCRIPTION

Converts integer `value` to hexadecimal number and puts result into `buf`.

### PARAMETERS

<code>value</code>	16-bit number to convert
<code>buf</code>	Character string of converted number

### RETURN VALUE

Pointer to end (null terminator) of string in `buf`.

### LIBRARY

STDIO.LIB

### SEE ALSO

`itoa`, `utoa`, `ltoa`



---

---

## IntervalMs

---

---

```
int IntervalMs( long ms );
```

### DESCRIPTION

Similar to `DelayMs` but provides a periodic delay based on the time from the previous call. Intended for use with `waitfor`.

### PARAMETERS

**ms**                      The number of milliseconds to wait.

### RETURN VALUE

0: Not finished.  
1: Delay has expired.

### LIBRARY

`COSTATE.LIB`

---

---

## IntervalSec

---

---

```
int IntervalSec( long sec );
```

### DESCRIPTION

Similar to `DelayMs` but provides a periodic delay based on the time from the previous call. Intended for use with `waitfor`.

### PARAMETERS

**sec**                      The number of seconds to delay.

### RETURN VALUE

0: Not finished.  
1: Delay has expired.

### LIBRARY

`COSTATE.LIB`

---

---

## IntervalTick

---

---

```
int IntervalTick( long tick );
```

### DESCRIPTION

Provides a periodic delay based on the time from the previous call. Intended for use with `waitfor`. A tick is 1/1024 seconds.

### PARAMETERS

**tick**                    The number of ticks to delay

### RETURN VALUE

0: Not finished.  
1: Delay has expired.

### LIBRARY

`COSTATE.LIB`

---

---

## ipres

---

---

```
void ipres( void );
```

### DESCRIPTION

Dynamic C expands this call inline. Restore previous interrupt priority by rotating the IP register.

### LIBRARY

`UTIL.LIB`

### SEE ALSO

`ipset`

---

---

## **ipset**

---

---

```
void ipset( int priority );
```

### **DESCRIPTION**

Dynamic C expands this call inline. Replaces current interrupt priority with another by rotating the new priority into the IP register.

### **PARAMETERS**

**priority**      Interrupt priority range 0–3, lowest to highest priority.

### **LIBRARY**

UTIL.LIB

### **SEE ALSO**

ipres

---

---

## **isalnum**

---

---

```
int isalnum( int c );
```

### **DESCRIPTION**

Tests for an alphabetic or numeric character, (A to Z, a to z and 0 to 9).

### **PARAMETERS**

**c**              Character to test.

### **RETURN VALUE**

0 if not an alphabetic or numeric character.  
!0 otherwise.

### **LIBRARY**

STRING.LIB

### **SEE ALSO**

isalpha, isdigit, ispunct

---

---

## isalpha

---

---

```
int isalpha( int c );
```

### DESCRIPTION

Tests for an alphabetic character, (A to Z, or a to z).

### PARAMETERS

**c**                    Character to test.

### RETURN VALUE

0 if not a alphabetic character.  
!0 otherwise.

### LIBRARY

STRING.LIB

### SEE ALSO

isalnum, isdigit, ispunct

---

---

## iscntrl

---

---

```
int iscntrl( int c );
```

### DESCRIPTION

Tests for a control character:  $0 \leq c \leq 31$  or  $c == 127$ .

### PARAMETERS

**c**                    Character to test.

### RETURN VALUE

0 if not a control character.  
!0 otherwise.

### LIBRARY

STRING.LIB

### SEE ALSO

isalpha, isalnum, isdigit, ispunct

---

---

## isCoDone

---

---

```
int isCoDone( CoData * p );
```

### DESCRIPTION

Determine if costatement is initialized and not running.

### PARAMETERS

**p**                      Address of costatement

### RETURN VALUE

1: Costatement is initialized and not running.  
0: Otherwise.

### LIBRARY

COSTATE.LIB

---

---

## isCoRunning

---

---

```
int isCoRunning( CoData * p );
```

### DESCRIPTION

Determine if costatement is stopped or running.

### PARAMETERS

**p**                      Address of costatement.

### RETURN VALUE

1 if costatement is running.  
0 otherwise.

### LIBRARY

COSTATE.LIB

---

---

## `isdigit`

---

---

```
int isdigit( int c );
```

### **DESCRIPTION**

Tests for a decimal digit: 0 - 9

### **PARAMETERS**

`c`                      Character to test.

### **RETURN VALUE**

0 if not a decimal digit.  
!0 otherwise.

### **LIBRARY**

`STRING.LIB`

### **SEE ALSO**

`isxdigit`, `isalpha`, `isalpha`

---

---

## isgraph

---

---

```
int isgraph( int c );
```

### DESCRIPTION

Tests for a printing character other than a space:  $33 \leq c \leq 126$

### PARAMETERS

**c**                      Character to test.

### RETURN VALUE

0: c is not a printing character.  
!0: c is a printing character.

### LIBRARY

STRING.LIB

### SEE ALSO

isprint, isalpha, isalnum, isdigit, ispunct

---

---

## islower

---

---

```
int islower( int c );
```

### DESCRIPTION

Tests for lower case character.

### PARAMETERS

**c**                      Character to test.

### RETURN VALUE

0 if not a lower case character.  
!0 otherwise.

### LIBRARY

STRING.LIB

### SEE ALSO

tolower, toupper, isupper

---

---

## isspace

---

---

```
int isspace( int c );
```

### DESCRIPTION

Tests for a white space, character, tab, return, newline, vertical tab, form feed, and space:  $9 \leq c \leq 13$  and  $c == 32$ .

### PARAMETERS

**c** Character to test.

### RETURN VALUE

0 if not, !0 otherwise.

### LIBRARY

STRING.LIB

### SEE ALSO

ispunct

---

---

## isprint

---

---

```
int isprint( int c );
```

### DESCRIPTION

Tests for printing character, including space:  $32 \leq c \leq 126$

### PARAMETERS

**c** Character to test.

### RETURN VALUE

0 if not a printing character, !0 otherwise.

### LIBRARY

STRING.LIB

### SEE ALSO

isdigit, isxdigit, isalpha, ispunct, isspace, isalnum, isgraph



---

---

## ispunct

---

---

```
int ispunct( int c );
```

### DESCRIPTION

Tests for a punctuation character.

Character	Decimal Code
space	32
! " # \$ % & ' ( ) * + , - . /	33 <= c <= 47
: ; < = > ? @	58 <= c <= 64
[ \ ] ^ _ `	91 <= c <= 96
{ } ~	123 <= c <= 126

### PARAMETERS

**c** Character to test.

### RETURN VALUE

0: Not a character.

!0: Is a character.

### LIBRARY

STRING.LIB

### SEE ALSO

isspace

---

---

## isupper

---

---

```
int isupper( int c );
```

### DESCRIPTION

Tests for upper case character.

### PARAMETERS

**c**                      Character to test.

### RETURN VALUE

0: Is not an uppercase character.  
!0: Is an uppercase character.

### LIBRARY

STRING.LIB

### SEE ALSO

tolower, toupper, islower

---

---

## isxdigit

---

---

```
int isxdigit( int c );
```

### DESCRIPTION

Tests for a hexadecimal digit: 0 - 9, A - F, a - f

### PARAMETERS

**c**                      Character to test.

### RETURN VALUE

0: Not a hexadecimal digit.  
!0: Is a hexadecimal digit.

### LIBRARY

STRING.LIB

### SEE ALSO

isdigit, isalpha, isalpha

---

---

## itoa

---

---

```
char * itoa( int value, char * buf );
```

### DESCRIPTION

Places up to a 5-digit character string, with a minus sign in the leftmost digit when appropriate, at `*buf`. The string represents `value`, a signed number.

Leading zeros are suppressed in the character string, except for one zero digit when `value = 0`. The longest possible string is “-32768.”

### PARAMETERS

<code>value</code>	16-bit signed number to convert
<code>buf</code>	Character string of converted number in base 10

### RETURN VALUE

Pointer to the end (null terminator) of the string in `buf`.

### LIBRARY

STDIO.LIB

### SEE ALSO

`atoi`, `utoa`, `ltoa`

---

---

## `i2c_check_ack`

---

---

```
int i2c_check_ack( void );
```

### **DESCRIPTION**

Checks if slave pulls data low for ACK on clock pulse. Allows for clocks stretching on SCL going high.

### **RETURN VALUE**

0: ACK sent from slave.  
1: NAK sent from slave.  
-1: Timeout occurred.

### **LIBRARY**

I2C.LIB

### **SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## `i2c_init`

---

---

```
void i2c_init( void );
```

### DESCRIPTION

Sets up the SCL and SDA port pins for open-drain output.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## `i2c_read_char`

---

---

```
int i2c_read_char( char * ch );
```

### DESCRIPTION

Reads 8 bits from the slave. Allows for clocks stretching on all SCL going high. This is not in the protocol for I<sup>2</sup>C, but allows I<sup>2</sup>C slaves to be implemented on slower devices.

### PARAMETERS

**ch**                    A one character return buffer.

### RETURN VALUE

0: Success.  
-1: Clock stretching timeout.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## i2c\_send\_ack

---

---

```
int i2c_send_ack( void );
```

### DESCRIPTION

Sends ACK sequence to slave. ACK is usually sent after a successful transfer, where more bytes are going to be read.

### RETURN VALUE

0: Success.  
-1: Clock stretching timeout.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## i2c\_send\_nak

---

---

```
int i2c_send_nak( void );
```

### DESCRIPTION

Sends NAK sequence to slave. NAK is often sent when the transfer is finished.

### RETURN VALUE

0: Success.  
-1: Clock stretching timeout.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## `i2c_start_tx`

---

---

```
int i2c_start_tx( void );
```

### **DESCRIPTION**

Initiates I<sup>2</sup>C transmission by sending the start sequence, which is defined as a high to low transition on SDA while SCL is high. The point being that SDA is supposed to remain stable while SCL is high. If it does not, then that indicates a start (S) or stop (P) condition. This function first waits for possible clock stretching, which is when a bus peripheral holds SCK low.

### **RETURN VALUE**

0: Success.  
-1: Clock stretching timeout.

### **LIBRARY**

I2C.LIB

### **SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## `i2c_startw_tx`

---

---

```
int i2c_startw_tx( void );
```

### DESCRIPTION

Initiates I<sup>2</sup>C transmission by sending the start sequence, which is defined as a high to low transition on SDA while SCL is high. The point being that SDA is supposed to remain stable while SCL is high. If it does not, then that indicates a start (S) or stop (P) condition. This function first waits for possible clock stretching, which is when a bus peripheral holds SCK low.

This function is essentially the same as `i2c_start_tx()` with the addition of a clock stretch delay, which is 2000 “counts,” inserted after the start sequence. (A count is an iteration through a loop.)

### RETURN VALUE

0: Success.  
-1: Clock stretching timeout.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.



---

---

## `i2c_stop_tx`

---

---

```
void i2c_stop_tx( void );
```

### DESCRIPTION

Sends the stop sequence to the slave, which is defined as bringing SDA high while SCL is high, i.e., the clock goes high, then data goes high.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## `i2c_write_char`

---

---

```
int i2c_write_char( char d );
```

### DESCRIPTION

Sends 8 bits to slave. Checks if slave pulls data low for ACK on clock pulse. Allows for clocks stretching on SCL going high.

### PARAMETERS

`d`                      Character to send

### RETURN VALUE

0: Success.  
-1: Clock stretching timeout.  
1: NAK sent from slave.

### LIBRARY

I2C.LIB

### SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

---

## kbhit

---

---

```
int kbhit( void );
```

### DESCRIPTION

Detects keystrokes in the Dynamic C Stdio window.

### RETURN VALUE

! 0 if a key has been pressed, 0 otherwise.

### LIBRARY

UTIL.LIB

---

---

## labs

---

---

```
long labs( long x );
```

### DESCRIPTION

Computes the long integer absolute value of long integer x.

### PARAMETERS

**x**                      Number to compute.

### RETURN VALUE

x, if  $x \geq 0$ .  
-x, otherwise.

### LIBRARY

MATH.LIB

### SEE ALSO

abs, fabs

---

---

## ldexp

---

---

```
float ldexp( float x, int n );
```

### DESCRIPTION

Computes  $x * (2^n)$ .

### PARAMETERS

<b>x</b>	The value between 0.5 inclusive, and 1.0
<b>n</b>	An integer

### RETURN VALUE

The result of  $x * (2^n)$ .

### LIBRARY

MATH.LIB

### SEE ALSO

frexp, exp

---

---

## log

---

---

```
float log( float x );
```

### DESCRIPTION

Computes the logarithm, base e, of real float value x.

### PARAMETERS

<b>x</b>	Float value
----------	-------------

### RETURN VALUE

The function returns  $-\text{INF}$  and signals a domain error when  $x \leq 0$ .

### LIBRARY

MATH.LIB

### SEE ALSO

exp, log10

---

---

## log\_clean

---

---

```
int log_clean( LogDest ld );
```

### DESCRIPTION

Reset only the specified destination class and stream (encoded as a LogDest value). This is only applicable to filesystem or XMEM destinations since they are locally persistent storage. XMEM is automatically cleaned at start-up time, since it is not assumed to be non-volatile.

If this operation is not applicable, 0 is returned with no further action.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETER

**ld** Destination class and stream. Use one of the constants `LOG_DEST_FS2` or `LOG_DEST_XMEM`, then OR in the stream number (0-63).

### RETURN VALUE

0: success  
-2: The stream is out-of-range for the class.

### LIBRARY

`log.lib`

---

---

## log\_close

---

---

```
int log_close( LogDestClass ldc );
```

### DESCRIPTION

Close the specified class, enumerating all streams. If the destination class is already closed, returns success.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETER

**ldc** Destination class. Use one of the constants `LOG_DEST_FS2`, `LOG_DEST_XMEM`, `LOG_DEST_UDP` or `LOG_DEST_ALL`. The latter case closes all open destinations.

### RETURN VALUE

0: success

### LIBRARY

`log.lib`

---

---

## log\_condition

---

---

```
int log_condition( LogDest ldst );
```

### DESCRIPTION

Return the state of the specified log destination. Destination classes or streams that are not configured cause a -2 return code.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETER

**ldst** Destination class and stream. Use one of the constants `LOG_DEST_FS2` or `LOG_DEST_XMEM`, then OR in the stream number (0-63).

### RETURN VALUE

0: Destination not open  
1: destination OK  
2: destination reached limit of its space quota  
-1: error in destination.  
-2: destination not configured

### LIBRARY

`log.lib`

---

---

## log\_format

---

---

```
char * log_format( LogEntry *le, char *buffer, int length, int pfx );
```

### DESCRIPTION

Given the log entry returned by `log_next()` or `log_prev()`, format the entry as an ASCII string. The string is constructed in Unix "syslog" format:

```
<%d>%.15s %.8s[%d]: %s
```

where the substitutions are:

- `%d`: facility/priority as decimal number (0-255)
- `%.15s`: date/time as "Mon dd hh:mm:ss"
- `%s`: process name - taken from `LOG_UDP_PNAME(0)` if defined, else "" (empty).
- `%d`: process ID, but the entry serial number is used instead.
- `%s`: the log entry data.

A null terminator is always added at `buffer[length-1]`, or at the end of the string if it fits in the buffer. If `pfx` is zero, then the above syslog prefix is not generated.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETERS

<b>le</b>	Log entry result from <code>log_next/log_prev()</code> .
<b>buffer</b>	Storage for result. Must be dimensioned at least 'length'.
<b>length</b>	Length of buffer. For the maximum sized log entry, the buffer should be 158 bytes. The minimum length must be greater than or equal to 43 (if <code>pfx</code> true) else 1. If a bad length is passed, the function returns without writing to buffer.
<b>pfx</b>	0: message text only; do not generate syslog prefix. 1: prefix plus message text. 2: prefix only (up to ']', then null terminator).

### RETURN VALUE

buffer address, or NULL if bad length passed.

### LIBRARY

`log.lib`

### SEE ALSO

`log_next`, `log_prev`

---

---

## log\_map

---

---

```
uint32 log_map( LogFacPri lfp );
```

### DESCRIPTION

Return the log destination class and stream, for a given facility/priority code. The result is up to four destinations packed into a longword. This function merely invokes the macro `LOG_MAP()`, which may be overridden by the application, but defaults to just the filesystem.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETER

**lfp** Facility/priority code. This is a single-byte code specified whenever any log message is added. Facility is coded in the 5 MSBs, and priority in the 3 LSBs.

### RETURN VALUE

Up to four destinations for a message of the specified facility and priority. Each byte in the resulting long word represents a destination/stream. A zero byte indicates no destination. If the result is all zeros, then a message of this type would be discarded.

### LIBRARY

`log.lib`



---

---

## log\_next

---

---

```
int log_next( LogDest ldst, LogEntry * le );
```

### DESCRIPTION

Retrieve next log entry. You must call `log_seek()` before calling this function the first time. Retrieval of stored log messages proceeds, for example, as follows:

```
log_seek(ldst, 0);           // seek to start
log_next(ldst, &L);         // get 1st entry
log_next(ldst, &L);         // get 2nd entry
log_prev(ldst, &L);         // get 2nd entry again
log_prev(ldst, &L);         // get 1st entry
log_prev(ldst, &L);         // returns -1
```

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETERS

<b>ldst</b>	Destination class and stream. Use one of the constants <code>LOG_DEST_FS2</code> or <code>LOG_DEST_XMEM</code> , then OR in the stream number (0-63).
<b>le</b>	Storage for result.

### RETURN VALUE

non-negative: length of log entry data  
-1: End of log or not open  
-2: Not a readable log destination class

### LIBRARY

`log.lib`

### SEE ALSO

`log_seek`, `log_prev`

---

---

## log\_open

---

---

```
int log_open( LogDestClass ldc, int clean );
```

### DESCRIPTION

Open the specified logging destination class. If necessary, this enumerates all possible streams within the class, opening them all (necessary only for FS2 class, since each file needs to be opened). Class LOG\_DEST\_ALL opens all configured classes.

If clean is true, then the dest is set to empty log, if that makes sense for the class.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETERS

<b>ldc</b>	Destination class: LOG_DEST_FS2, LOG_DEST_UDP, LOG_DEST_XMEM or LOG_DEST_ALL.
<b>clean</b>	Boolean, should the destination be erased before using?

### RETURN VALUE

0: success  
-1: unknown LogDestClass value

### LIBRARY

`log.lib`

---

---

## log\_prev

---

---

```
int log_prev( LogDest ldst, LogEntry * le );
```

### DESCRIPTION

Retrieve previous log entry. You must call `log_seek()` before calling this function the first time. Retrieval of stored log messages proceeds, for example, as follows:

```
log_seek(ldst, 1);           // seek to end
log_prev(ldst, &L);         // get last entry
log_prev(ldst, &L);         // get 2nd last entry
log_next(ldst, &L);         // get 2nd last entry again
log_next(ldst, &L);         // get last entry
log_next(ldst, &L);         // returns -1
```

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETERS

<b>ldst</b>	Destination class and stream. Use one of the constants <code>LOG_DEST_FS2</code> or <code>LOG_DEST_XMEM</code> , then OR in the stream number (0-63).
<b>le</b>	Storage for result.

### RETURN VALUE

non-negative = length of log entry data  
-1 = Start of log or not open  
-2 = Not a readable log destination class

### LIBRARY

`log.lib`

### SEE ALSO

`log_seek`, `log_next`

---

---

## log\_put

---

---

```
int log_put( LogFacPri ifp, uint8 fmt, const char *data, int length );
```

### DESCRIPTION

Add a log entry. The specified facility/priority is mapped to the appropriate destination(s), as configured by the macros. If the destination exists, then the log entry is added; otherwise, the entry is quietly ignored. If a destination is unable to fit the log entry, and the destination is configured as “circular,” then the first few entries may be deleted to make room. If this cannot be done, or an unrecoverable error occurs, then -2 is returned. For non-circular destinations, -2 is returned when it becomes full.

Since multiple log destinations can result from the given facility/priority, it can be difficult to determine which actual destination caused an error. You can use the `log_map()` function to determine the destinations, then check each destination's state using `log_condition()`.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETERS

<b>ifp</b>	Facility/priority code. Facility in 5 MSBs, priority in 3 LSBs.
<b>fmt</b>	Format code. 0 for ascii string, others user-defined.
<b>data</b>	Pointer to first byte of data to store.
<b>length</b>	Length of data. Must be between 0 and 115 ( <code>LOG_MAX_MESSAGE</code> ) inclusive.

### RETURN VALUE

0 = success  
-1 = Message too long (over 115).  
-2 = Unrecoverable error in destination. This return code usually means that the destination is unusable and further entries for that destination will probably meet the same fate. This can also mean that the destination has not been opened.

### LIBRARY

`log.lib`

---

---

## log\_seek

---

---

```
int log_seek( LogDest ldst, int );
```

### DESCRIPTION

Position log for readback. The next call to `log_next()` will return the first entry in the log (if `whence=0`), or `log_prev()` will return the last entry (if `whence=1`).

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

### PARAMETERS

<b>ldst</b>	Destination class and stream. Use one of the constants <code>LOG_DEST_FS2</code> or <code>LOG_DEST_XMEM</code> , then OR in the stream number (0-63).
<b>whence</b>	0: first entry. 1: last entry. other values reserved.

### RETURN VALUE

0 = success.  
-1 = Log empty.  
-2 = Unrecoverable error or not open.  
-3 = Not a seekable or configured log destination class.  
-4 = invalid whence parameter.

### LIBRARY

`log.lib`

### SEE ALSO

`log_next`, `log_prev`

---

---

## log10

---

---

```
float log10( float x );
```

### DESCRIPTION

Computes the base 10 logarithm of real `float` value `x`.

### PARAMETERS

**x**                    Value to compute

### RETURN VALUE

The log base 10 of `x`.

The function returns `-INF` and signals a domain error when  $x \leq 0$ .

### LIBRARY

`MATH.LIB`

### SEE ALSO

`log`, `exp`

---

---

## longjmp

---

---

```
void longjmp( jmp_buf env, int val );
```

### DESCRIPTION

Restores the stack environment saved in array `env [ ]`. See the description of `setjmp()` for details of use.

**Note:** you cannot use `longjmp()` to move out of slice statements, costatements, or cofunctions.

### PARAMETERS

**env**                    Environment previously saved with `setjmp()`.

**val**                    Integer result of `setjmp()`.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`setjmp`

---

---

## loophead

---

---

```
void loophead( void );
```

### DESCRIPTION

This function should be called within the main loop in a program. It is necessary for proper single-user cofunction abandonment handling.

When two costatements are requesting access to a single-user cofunction, the first request is honored and the second request is held. When `loophead()` notices that the first caller is not being called each time around the loop, it cancels the request, calls the abandonment code and allows the second caller in.

See `Samples\Cofunc\Cofaband.c` for sample code showing abandonment handling.

### LIBRARY

COFUNC.LIB

---

---

## loopinit

---

---

```
void loopinit( void );
```

### DESCRIPTION

This function should be called in the beginning of a program that uses single-user cofunctions. It initializes internal data structures that are used by `loophead()`.

### LIBRARY

COFUNC.LIB

---

---

## lsqrt

---

---

```
unsigned int lsqrt( unsigned long x );
```

### DESCRIPTION

Computes the square root of *x*. Note that the return value is an unsigned int. The fractional portion of the result is truncated.

### PARAMETERS

**x**                      long int input for square root computation

### RETURN VALUE

Square root of *x* (fractional portion truncated).

### LIBRARY

MATH.LIB

---

---

## ltoa

---

---

```
char * ltoa( long num, char * ibuf )
```

### DESCRIPTION

This function outputs a signed long number to the character array.

### PARAMETERS

**num**                      Signed long number.

**ibuf**                      Pointer to character array.

### RETURN VALUE

Pointer to the same array passed in to hold the result.

### LIBRARY

STDIO.LIB

### SEE ALSO

ltoa



---

---

## ltoan

---

---

```
int ltoan( long num );
```

### DESCRIPTION

This function returns the number of characters required to display a signed long number.

### PARAMETERS

**num**                    32-bit signed number.

### RETURN VALUE

The number of characters to display signed long number.

### LIBRARY

STDIO.LIB

### SEE ALSO

ltoa

---

---

## lx\_format

---

---

```
int lx_format( FSLXnum lxn, long wearlevel );
```

### DESCRIPTION

Format a specified file system extent. This must not be called before calling `fs_init()`. All files which have either or both metadata and data on this extent are deleted. Formatting can be quite slow (depending on hardware) so it is best performed after power-up, if at all.

### PARAMETERS

<b>lxn</b>	Logical extent number (1.. <code>_fs.num_lx</code> inclusive).
<b>wearlevel</b>	Initial wearlevel value. This should be 1 if you have a new flash, and some larger number if the flash is used. If you are reformatting a flash, you can use 0 to use the old flash wear levels.

### RETURN VALUE

0: Success.  
!0: Failure.

### ERRNO VALUES

ENODEV - no such extent number, or extent is reserved.  
EBUSY - one or more files were open on this extent.  
EIO - I/O error during format. If this occurs, retry the format operation. If it fails again, there is probably a hardware error.

### LIBRARY

FS2.LIB

### SEE ALSO

`fs_init`, `fs_format`

---

---

## mbr\_CreatePartition

---

---

```
int mbr_CreatePartition( mbr_drive *drive, int pnum, char type );
```

### DESCRIPTION

Creates or modifies the partition specified. The partition being modified must not be mounted, and should be released by filesystem use (that is, its `fs_part` pointer must be null). The new partition values should be placed in the appropriate partition structure within the drive structure. For example,

```
drive.part [partnum].bootflag = 0;
drive.part [partnum].starthead = 0xfe;
drive.part [partnum].startseccyl = 0;
drive.part [partnum].parttype = 0xda;
drive.part [partnum].endhead = 0xfe;
drive.part [partnum].endseccyl = 0;
drive.part [partnum].startsector = start;
drive.part [partnum].partseccyl = ((PART_SZ) / 512) + 1;
mbr_CreatePartition(&drive, partnum, 0xda);
```

For more information on the partition structure (`mbr_part`) look in `part_defs.lib`.

The `type` parameter should match the type as it currently exists on the drive, unless this is unused. Some values for the `type` parameter are already in use. A list of known partition types is at:

[www.win.tue.nl/~aeb/partitions/partition\\_types-1.html](http://www.win.tue.nl/~aeb/partitions/partition_types-1.html)

**Note:** Starting with Dynamic C 9.01, this function BLOCKS!

### PARAMETERS

<b>drive</b>	Pointer to a MBR drive structure
<b>pnum</b>	Partition number to be created or modified
<b>type</b>	Type that exists on the physical drive partition now

### RETURN VALUE

0 for success

- EIO for Error trying to read drive/device or structures.
- EINVAL if drive structure, pnum or type is invalid.
- EPERM if the partition has not been enumerated or is currently mounted.
- EUNFORMAT if the drive is accessible, but not formatted.
- EBUSY if the device is busy. (Valid prior to Dynamic C 9.01)

### LIBRARY

PART.LIB

---

---

## mbr\_EnumDevice

---

---

```
mbr_EnumDevice( mbr_drvr *driver, mbr_dev *dev, int devnum, int
                (*checktype)() );
```

### DESCRIPTION

This routine is called to learn about devices present on the driver passed in. The device will be added to the linked list of enumerated devices. Partition information will be filled in from the master boot record (MBR). Pointers to file system level partition information structures will be set to NULL.

### PARAMETERS

<b>driver</b>	Pointer to a DOS controller structure (setup during init of storage device driver.)
<b>dev</b>	Pointer to a drive structure to be filled in.
<b>devnum</b>	Physical device number of device on the driver.
<b>checktype</b>	Routine that takes an unsigned char partition type and returns 1 if of sought type and zero if not. Pass NULL for this parameter to bypass this check.

### RETURN VALUE

0 for success  
-EIO for Error trying to read the device or structure.  
-EINVAL if devnum invalid or does not exist.  
-ENOMEM if memory for page buffer is not available.  
-EUNFORMAT if the device is accessible, but not formatted. You can use it provided it is formatted/partitioned by either this library or another system.  
-EBADPART if the partition table on the device is invalid  
-ENOPART if the device does not have any sought partitions, If checktype parameter is NULL, this test is bypassed. This code is superseded by any other error detected.  
-EXIST if the device has already been enumerated.  
-EBUSY if the device is busy.

### LIBRARY

PART.LIB

---

---

## mbr\_FormatDevice

---

---

```
int mbr_FormatDevice( mbr_dev * dev );
```

### DESCRIPTION

Creates or rewrites the Master Boot Record on the device given. The routine will only rewrite the Boot Loader code if an MBR already exists on the device. The existing partition table will be preserved. To modify an existing partition table use `mbr_CreatePartion`.

**Note:** This routine is NOT PROTECTED from power loss and can make existing partitions inaccessible if interrupted.

**Note:** This function is BLOCKING.

### PARAMETERS

**dev**                      Pointer to MBR device structure

### RETURN VALUE

- 0 for success.
- EEXIST if the MBR exists, writing Boot Loader only
- EIO for Error trying to read the device or structure
- EINVAL if the Device structure is not valid
- ENOMEM if memory for page buffer is not available
- EPERM if drive has mounted or FS enumerated partition(s)

### LIBRARY

PART.LIB

---

---

## mbr\_MountPartition

---

---

```
int mbr_MountPartition( mbr_drive * drive, int pnum );
```

### DESCRIPTION

Marks the partition as mounted. It is the higher level codes responsibility to verify that the `fs_part` pointer for a partition is not in use (null) as this would indicate that another system is in the process of mounting this device.

### PARAMETERS

<code>drive</code>	Pointer to a drive structure
<code>pnum</code>	Partition number to be mounted

### RETURN VALUE

0 for success  
- EINVAL if Drive or Partition structure or pnum is invalid.  
- ENOPART if Partition does not exist on the device.

### LIBRARY

PART.LIB

---

---

## mbr\_UnmountPartition

---

---

```
int mbr_UnmountPartition( mbr_drive * drive, int pnum );
```

### DESCRIPTION

Marks the partition as unmounted. The partition must not have any user partition data attached (through mounting at a higher level). If the `fs_part` pointer for the partition being unmounted is not null, an `EPERM` error is returned.

### PARAMETERS

<code>drive</code>	Pointer to a drive structure containing the partition
<code>pnum</code>	Partition number to be unmounted

### RETURN VALUE

0 for success  
- `EINVAL` if the Drive structure or `pnum` is invalid.  
- `ENOPART` if the partition is enumerated at a higher level.

### LIBRARY

`PART.LIB`

---

---

## `mbr_ValidatePartitions`

---

---

```
int mbr_ValidatePartitions( mbr_drive * drive );
```

### DESCRIPTION

This routine will validate the partition table contained in the drive structure passed. It will verify that all partitions fit within the bounds of the drive and that no partitions overlap.

### PARAMETERS

`drive`            Pointer to a drive structure

### RETURN VALUE

0 for success  
-EINVAL if the partition table in the drive structure is invalid.

### LIBRARY

PART.LIB



---

---

## md5\_append

---

---

```
void md5_append( md5_state_t * pms, char * data, int nbytes );
```

### DESCRIPTION

This function will take a buffer and compute the MD5 hash of its contents, combined with all previous data passed to it. This function can be called several times to generate the hash of a large amount of data.

### PARAMETERS

<b>md5_append</b>	Pointer to the <code>md5_state_t</code> structure that was initialized by <code>md5_init</code> .
<b>data</b>	Pointer to the data to be hashed.
<b>nbytes</b>	Length of the data to be hashed.

### LIBRARY

MD5.LIB

---

---

## md5\_init

---

---

```
void md5_init( md5_state_t * pms );
```

### DESCRIPTION

Initialize the MD5 hash process. Initial values are generated for the structure, and this structure will identify a particular transaction in all subsequent calls to the md5 library.

### PARAMETER

<b>pms</b>	Pointer to the <code>md5_state_t</code> structure.
------------	--

### LIBRARY

MD5.LIB

---

---

## md5\_finish

---

---

```
void md5_finish( md5_state_t * pms, char digest[16] );
```

### DESCRIPTION

Completes the hash of all the received data and generates the final hash value.

### PARAMETERS

<b>pms</b>	Pointer to the <code>md5_state_t</code> structure that was initialized by <code>md5_init</code> .
<b>digest</b>	The 16-byte array that the hash value will be written into.

### LIBRARY

MD5.LIB

---

---

## memchr

---

---

**NEAR SYNTAX:** `void * _n_memchr( void * src, int ch, unsigned int n );`  
**FAR SYNTAX:** `void far * _f_memchr( void far * src, int ch, size_t n );`

**Note:** By default, `memchr()` is defined to `_n_memchr()`.

### DESCRIPTION

Searches up to `n` characters at memory pointed to by `src` for character `ch`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<code>src</code>	Pointer to memory source.
<code>ch</code>	Character to search for.
<code>n</code>	Number of bytes to search.

### RETURN VALUE

Pointer to first occurrence of `ch` if found within `n` characters. Otherwise returns null.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strrchr`, `strstr`

---

---

## memcmp

---

---

**NEAR SYNTAX:** `int _n_memcmp( void *s1, void *s2, size_t n );`  
**FAR SYNTAX:** `int _f_memcmp( void far *s1, void far *s2, size_t n );`

**Note:** By default, `memcmp()` is defined to `_n_memcmp()`.

### DESCRIPTION

Performs unsigned character by character comparison of two memory blocks of length `n`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>s1</b>	Pointer to block 1.
<b>s2</b>	Pointer to block 2.
<b>n</b>	Maximum number of bytes to compare.

### RETURN VALUE

<0: A character in `str1` is less than the corresponding character in `str2`.  
0: `str1` is identical to `str2`.  
>0: A character in `str1` is greater than the corresponding character in `str2`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strncmp`

---

---

## memcpy

---

---

**NEAR SYNTAX:** `void *_n_memcpy( void *dst, void *src, unsigned int n );`  
**FAR SYNTAX:** `void far *_f_memcpy( void far *dst, void far *src, size_t n );`

**Note:** By default, `memcpy()` is defined to `_n_memcpy()`.

### DESCRIPTION

Copies a block of bytes from one destination to another. Overlap is handled correctly.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>dst</b>	Pointer to memory destination
<b>src</b>	Pointer to memory source
<b>n</b>	Number of characters to copy

### RETURN VALUE

Pointer to destination.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`memmove`, `memset`

---

---

## memmove

---

---

**NEAR SYNTAX:** `void *_n_memmove( void *dst, void *src, unsigned int n );`

**FAR SYNTAX:** `_f_memmove( void far * dst, void far * src, size_t n);`

**Note:** By default `memmove()` is defined to `_n_memmove()`.

### DESCRIPTION

Copies a block of bytes from one destination to another. Overlap is handled correctly.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>dst</b>	Pointer to memory destination
<b>src</b>	Pointer to memory source
<b>n</b>	Number of characters to copy

### RETURN VALUE

Pointer to destination.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`memcpy`, `memset`

---

---

## memset

---

---

**NEAR SYNTAX:** `void * _n_memset( void * dst, int chr, unsigned int n );`  
**FAR SYNTAX:** `void far * _f_memset( void far * dst, int chr, size_t n );`

**Note:** By default, `memset()` is defined to `_n_memset()`.

### DESCRIPTION

Sets the first `n` bytes of a block of memory pointed to by `dst` to the character `chr`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>dst</b>	Block of memory to set
<b>chr</b>	Character that will be written to memory
<b>n</b>	Amount of bytes to set

### RETURN VALUE

`dst`: Pointer to block of memory.

### LIBRARY

`STRING.LIB`

---

---

## mtime

---

---

```
unsigned long mktime( struct tm * timeptr );
```

### DESCRIPTION

Converts the contents of structure pointed to by `timeptr` into seconds.

```
struct tm {
    char tm_sec;           // seconds 0-59
    char tm_min;           // 0-59
    char tm_hour;          // 0-23
    char tm_mday;          // 1-31
    char tm_mon;           // 1-12
    char tm_year;          // 80-147 (1980-2047)
    char tm_wday;          // 0-6 0==sunday
};
```

### PARAMETERS

`timeptr`            Pointer to `tm` structure

### RETURN VALUE

Time in seconds since January 1, 1980.

### LIBRARY

RTCLIB.LIB

### SEE ALSO

`mktime`, `tm_r`, `tm_w`



---

---

## mktm

---

---

```
unsigned int mktm( struct tm * timeptr, unsigned long time );
```

### DESCRIPTION

Converts the seconds (*time*) to date and time and fills in the fields of the *tm* structure with the result.

```
struct tm {
    char tm_sec;           // seconds 0-59
    char tm_min;           // 0-59
    char tm_hour;          // 0-23
    char tm_mday;          // 1-31
    char tm_mon;           // 1-12
    char tm_year;          // 80-147 (1980-2047)
    char tm_wday;          // 0-6 0==sunday
};
```

### PARAMETERS

<b>timeptr</b>	Address to store date and time into structure:
<b>time</b>	Seconds since January 1, 1980.

### RETURN VALUE

0

### LIBRARY

RTCLIB

### SEE ALSO

`mkttime`, `tm_rd`, `tm_wr`

---

---

## modf

---

---

```
float modf( float x, int * n );
```

### DESCRIPTION

Splits  $x$  into a fraction and integer,  $f + n$ .

### PARAMETERS

<b>x</b>	Floating-point integer
<b>n</b>	An integer

### RETURN VALUE

The integer part in  $*n$  and the fractional part satisfies  $|f| < 1.0$

### LIBRARY

MATH.LIB

### SEE ALSO

fmod, ldexp

---

---

## nf\_eraseBlock

---

---

```
int nf_eraseBlock( nf_device * dev, long page );
```

### DESCRIPTION

Erases the block that contains the specified page on the specified NAND flash device. Check for completion of the erase operation using either `nf_isBusyRBHW()` or `nf_isBusyStatus()`.

Normally, this function will not allow a bad block to be erased. However, when `NFLASH_CANERASEBADBLOCKS` is defined by the application, the bad block check is not performed, and the application is allowed to erase any block, regardless of whether it is marked good or bad.

### PARAMETERS

<b>dev</b>	Pointer to an initialized <code>nf_device</code> structure
<b>page</b>	Page specifies the zero-based number of a NAND flash page in the block to be erased, relative to the first “good” page.

### RETURN VALUE

- 0: Success, or the first error result encountered
- 1: NAND flash device is busy
- 2: Block check time out error
- 3: Page is in a bad block

### LIBRARY

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### SEE ALSO

`CalculateECC256`, `ChkCorrectECC256`, `xCalculateECC256`,  
`xChkCorrectECC256`

---

---

## nf\_getPageCount

---

---

```
long nf_getPageCount( nf_device * dev );
```

### DESCRIPTION

Returns the number of program pages on the particular NAND flash device.

### PARAMETERS

**dev**                    Pointer to an `nf_device` structure for an initialized NAND flash device.

### RETURN VALUE

The number of program pages on the NAND flash device.

### LIBRARY

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### SEE ALSO

`CalculateECC256`, `ChkCorrectECC256`, `xCalculateECC256`,  
`xChkCorrectECC256`

---

---

## **nf\_getPageSize**

---

---

```
long nf_getPageSize( nf_device * dev );
```

### **DESCRIPTION**

Returns the size in bytes (excluding “spare” bytes) of each program page on the particular NAND flash device.

### **PARAMETERS**

**dev**                      Pointer to an `nf_device` structure for an initialized NAND flash device.

### **RETURN VALUE**

The number of data bytes in the NAND flash's program page, excluding the “spare” bytes used for ECC storage, etc.

### **LIBRARY**

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### **SEE ALSO**

`CalculateECC256`, `ChkCorrectECC256`, `xCalculateECC256`,  
`xChkCorrectECC256`

---

---

## nf\_initDevice

---

---

```
int nf_initDevice( nf_device * dev, int which );
```

### DESCRIPTION

Initializes a particular NAND flash device. This function must be called before the particular NAND flash device can be used. See `nf_devtable []` in `NFLASH.LIB` for the user-updatable list of supported NAND flash devices. Note that `xalloc` is called to allocate buffer(s) memory for each NAND flash device; a run time error will occur if the available `xmem` RAM is insufficient.

There are two modes of operation for NAND flash devices: FAT and direct. If you are using the FAT file system in the default configuration, i.e., the NAND flash has one FAT partition that takes up the entire device, you do not need to call `nf_initDevice()`. You only need to call `nf_InitDriver()`, which is the default device driver for the FAT file system on a NAND flash device.

Configurations other than the default one require more work. For example, having two partitions on the device, one a FAT partition and the other a non-FAT partition, require you to know how to fit more than one partition on a device. A good example of how to do this is in the remote application upload utility. The function `d1m_initserialflash()` in `/LIB/RCM3300/downloadmanager.lib` is where to look for code details. The upload utility is specifically for the RCM3300; however, even without the RCM3300, the utility is still useful in detailing what is necessary to manage multiple partitions.

The second mode of operation for NAND flash devices is direct access. An application that directly accesses the NAND flash (using calls such as `nf_readPage()` and `nf_writePage()`) may define `NFLASH_USEERASEBLOCKSIZE` to be either 0 (zero) or 1 (one) before `NFLASH.LIB` is #used, in order to set the NAND flash driver's main data program unit size to either the devices' program page size of 512 bytes or to its erase block size of 16 KB.

If not defined by the application, `NFLASH_USEERASEBLOCKSIZE` is set to the value 1 in `NFLASH.LIB`; this mode should maximize the NAND flash devices' life.

`NFLASH_USEERASEBLOCKSIZE` value 1 sets the driver up to program an erase block size at a time. This mode may be best for applications with only a few files open in write mode with larger blocks of data being written, and may be especially good at append operations. The trade off is reduced flash erasures at the expense of chunkier overhead due to the necessity of performing all 32 pages' ECC calculations for each programming unit written.

`NFLASH_USEERASEBLOCKSIZE` value 0 sets the driver up to program a program page size at a time. This mode may be best for applications with more than a few files open in write mode with smaller blocks of data being written, and may be especially good at interleaved file writes and/or random access write operations. The trade off is increased flash erasures with the benefit of spread out overhead due to the necessity of performing only 1 page's ECC calculations per programming unit written.

---

---

## `nf_initDevice (cont'd)`

---

---

### PARAMETERS

- |              |   |
|--------------|---|
| <b>dev</b>   | Pointer to an <code>nf_device</code> structure that will be filled in. An initialized <code>nf_device</code> struct acts as a handle for the NAND flash device. |
| <b>which</b> | Number of the NAND flash device to initialize. Currently supported device numbers are 0 for the soldered-on device or 1 for the socketed NAND flash device.     |

### RETURN VALUE

- 0: Success
- 1: Unknown index or bad internal I/O port information
- 2: Error communicating with flash chip
- 3: Unknown flash chip type

### LIBRARY

`NFLASH.LIB` (This function was introduced in Dynamic C 9.01)

### SEE ALSO

`CalculateECC256`, `ChkCorrectECC256`, `xCalculateECC256`,  
`xChkCorrectECC256`

---

---

## nf\_InitDriver

---

---

```
int nf_InitDriver( mbr_drvr * driver, void * device_list );
```

### DESCRIPTION

Initializes the NAND flash controller.

### PARAMETERS

<b>driver</b>	Empty <code>mbr_drvr</code> structure. It must be initialized with this function before it can be used with the FAT file system. More information on this structure can be found in the Dynamic C Module document titled, "FAT File System User's Manual," available on the <a href="#">Rabbit Semiconductor</a> website.
<b>device_list</b>	If not null, this is a pointer to the head of a linked list of <code>nf_device</code> structures for NAND flash devices that have each already been initialized by calling <code>nf_initDevice()</code> .  If <code>device_list</code> is null, then this function attempts to initialize all NAND flash devices and provide a default linked list of <code>nf_device</code> structures in order from device number 0 on up. If the initialization of a NAND flash device is unsuccessful, then its <code>nf_device</code> structure is not entered into the linked list.

### RETURN VALUE

0: Success  
<0: Negative value of a FAT file system error code

### LIBRARY

NFLASH\_FAT.LIB (This function was introduced in Dynamic C 9.01)



---

---

## `nf_isBusyRBHW`

---

---

```
int nf_isBusyRBHW( nf_device * dev );
```

### DESCRIPTION

Returns 1 if the specified NAND flash device is busy. Uses the hardware Ready/Busy check method, and can be used to determine the device's busy status even at the start of a read page command. Note that this function briefly enforces the Ready/Busy input port bit, reads the pin status, and then restores the port bit to its previous input/output state. There should be little or no visible disturbance of the LED output which shares the NAND flash's Ready/Busy status line.

### PARAMETERS

<b>dev</b>	Pointer to an initialized <code>nf_device</code> structure for the particular NAND flash chip.
------------	--

### RETURN VALUE

- 1: Busy
- 0: Ready, (not currently transferring a page to be read, or erasing or writing a page)
- 1: Error (unsupported Ready/Busy input port)

### LIBRARY

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### SEE ALSO

`nf_isBusyStatus`

---

---

## `nf_isBusyStatus`

---

---

```
int nf_isBusyStatus( nf_device * dev );
```

### DESCRIPTION

Returns 1 if the specified NAND flash device is busy erasing or writing to a page. Uses the software status check method, which can not (must not) be used to determine the device's busy status at the start of a read page command.

### PARAMETERS

<b>dev</b>	Pointer to an initialized <code>nf_device</code> structure for the particular NAND flash chip
------------	---

### RETURN VALUE

1: Busy  
0: Ready (not currently erasing or writing a page)

### LIBRARY

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### SEE ALSO

`nf_isBusyRBHW`

---

---

## nf\_readPage

---

---

```
int nf_readPage( nf_device * dev, long buffer, long page );
```

### DESCRIPTION

Reads data from the specified NAND flash device and page to the specified buffer in xmem. Note that in the case of most error results at least some of the NAND flash page's content has been read into the specified buffer. Although the buffer content must be considered unreliable, it can sometimes be useful for inspecting page content in “bad” blocks.

### PARAMETERS

<b>dev</b>	Pointer to an initialized <code>nf_device</code> structure
<b>buffer</b>	Physical address of the xmem buffer to read data into
<b>page</b>	Specifies the zero-based number of a NAND flash page to be read, relative to the first “good” page’s number.

### RETURN VALUE

- 0: Success, or the first error result encountered
- 1: NAND flash device is busy
- 2: Block check time out error
- 3: Page is in a bad block
- 4: Page read time out error
- 5: Uncorrectable data or ECC error

### LIBRARY

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### SEE ALSO

CalculateECC256, ChkCorrectECC256, xCalculateECC256, xChkCorrectECC256

---

---

## nf\_writePage

---

---

```
int nf_writePage( nf_device * dev, long buffer, long page );
```

### DESCRIPTION

Writes data to the specified NAND flash device and page from the specified buffer in xmem. Check for completion of the write operation using `nf_isBusyRBHW()` or `nf_isBusyStatus()`.

### PARAMETERS

<b>dev</b>	Pointer to an initialized <code>nf_device</code> structure
<b>buffer</b>	Physical address of the xmem data to be written
<b>page</b>	Specifies the zero-based number of a NAND flash page to be written, relative to the first “good” page.

### RETURN VALUE

- 0: Success, or the first error result encountered
- 1: NAND flash device is busy
- 2: Block check time out error
- 3: Page is in a bad block
- 4: XMEM/root memory transfer error
- 5: Erase block or program page operation error.

### LIBRARY

NFLASH.LIB (This function was introduced in Dynamic C 9.01)

### SEE ALSO

`CalculateECC256`, `ChkCorrectECC256`, `xCalculateECC256`,  
`xChkCorrectECC256`

---

---

## nf\_XD\_Detect

---

---

```
long nf_XD_Detect( int debounceMode );
```

### DESCRIPTION

This function attempts to read the xD card ID and searches the internal device table for that ID in detect mode 1. In detect mode 0 it just uses the xD card detect.

Assumes only one XD card present.

WARNING! - This should not be called to determine if it is safe to do write operations if there is a chance a removable device might be pulled between calling it and the write. It is best used to determine if a device is present to proceed with an automount after a device has been unmounted in SW and removed.

### PARAMETERS

**debounceMode**    0 - no debouncing  
                  1 - busy wait for debouncing interval  
                  2 - for use if function to be called until debouncing interval is done, e.g.,  
                      waitfor(rc = nf\_XD\_Detect(1) != -EAGAIN);  
                      - EAGAIN will be returned until done.

### RETURN VALUE

>0: The ID that was found on the device and in the table  
-EBUSY: NAND flash device is busy  
-ENODEV: No device found  
-EAGAIN: if debounceMode equals 2, then not done debouncing, try again

### LIBRARY

NFLASH\_FAT.LIB

---

---

## OpenInputCompressedFile

---

---

```
int OpenInputCompressedFile( ZFILE * ifp, long fn );
```

### DESCRIPTION

Opens a file for input. This function sets up the LZ compression algorithm window associated with the ZFILE file. The second parameter is the file handle (FS2) or address (`#zimport`) of the input file to be opened. If the file is already compressed, after calling this function the file can be decompressed by calling `ReadCompressedFile()`. If the file handle points to an uncompressed FS2 file, after calling this function the resulting ZFILE file can be compressed by calling `CompressFile()`.

The `INPUT_COMPRESSION_BUFFERS` macro controls the memory allocated by this function. It defaults to 1.

### PARAMETERS

<code>ifp</code>	ZFILE file descriptor
<code>fn</code>	Address or handle of input file

### RETURN VALUE

0: Failure  
1: Success

### LIBRARY

`LZSS.LIB`

### SEE ALSO

`CloseInputCompressedFile`, `CompressFile`, `ReadCompressedFile`

---

---

## OpenOutputCompressedFile

---

---

```
int OpenOutputCompressedFile( ZFILE * ofp, int fn );
```

### DESCRIPTION

Open an FS2 file for compressed output. This function sets up the LZ compression algorithm window and tree associated with the ZFILE file. The second parameter is the file handle (FS2) of the output file to be written to. Note that this MUST be an FS2 file handle, or the open will fail.

The OUTPUT\_COMPRESSION\_BUFFERS macro must be defined as a positive non-zero number if compression is being used.

### PARAMETERS

<code>ofp</code>	ZFILE file descriptor
<code>fn</code>	FS2 handle of output file

### RETURN VALUE

0: Failure  
1: Success

### LIBRARY

LZSS.LIB

### SEE ALSO

CloseOutputCompressedFile

---

---

## OS\_ENTER\_CRITICAL

---

---

```
void OS_ENTER_CRITICAL( void );
```

### DESCRIPTION

Enter a critical section. Interrupts will be disabled until `OS_EXIT_CRITICAL()` is called. Task switching is disabled. This function must be used with great care, since misuse can greatly increase the latency of your application. Note that nesting `OS_ENTER_CRITICAL()` calls will work correctly.

### LIBRARY

UCOS2.LIB

---

---

## OS\_EXIT\_CRITICAL

---

---

```
void OS_EXIT_CRITICAL( void );
```

### DESCRIPTION

Exit a critical section. If the corresponding previous `OS_ENTER_CRITICAL()` call disabled interrupts (that is, interrupts were not already disabled), then interrupts will be enabled. Otherwise, interrupts will remain disabled. Hence, nesting calls to `OS_ENTER_CRITICAL()` will work correctly.

### LIBRARY

UCOS2.LIB



---

---

## OSFlagAccept

---

---

```
OS_FLAGS OSFlagAccept( OS_FLAG_GRP * pgrp, OS_FLAGS flags, INT8U
    wait_type, INT8U * err );
```

### DESCRIPTION

This function is called to check the status of a combination of bits to be set or cleared in an event flag group. Your application can check for ANY bit to be set/cleared or ALL bits to be set/cleared.

This call does not block if the desired flags are not present.

### PARAMETERS

- |                  |   |
|------------------|---|
| <b>pgrp</b>      | Pointer to the desired event flag group.  |
| <b>flags</b>     | Bit pattern indicating which bit(s) (i.e. flags) you wish to check. E.g., if your application wants to wait for bits 0 and 1 then flags should be 0x03.   |
| <b>wait_type</b> | Specifies whether you are checking for ALL bits to be set/cleared or ANY of the bits to be set/cleared. You can specify the following argument: <ul style="list-style-type: none"><li>• OS_FLAG_WAIT_CLR_ALL - You will check ALL bits in flags to be clear (0)</li><li>• OS_FLAG_WAIT_CLR_ANY - You will check ANY bit in flags to be clear (0)</li><li>• OS_FLAG_WAIT_SET_ALL - You will check ALL bits in flags to be set (1)</li><li>• OS_FLAG_WAIT_SET_ANY - You will check ANY bit in flags to be set (1)</li></ul> |

**Note:** Add OS\_FLAG\_CONSUME if you want the event flag to be consumed by the call. Example, to wait for any flag in a group AND then clear the flags that are present, set the wait\_type parameter to:

```
OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME
```

---

---

## OSFlagAccept (cont'd)

---

---

**err** Pointer to an error code. Possible values are:

- OS\_NO\_ERR - No error
- OS\_ERR\_EVENT\_TYPE - Not pointing to an event flag group
- OS\_FLAG\_ERR\_WAIT\_TYPE - Proper wait\_type argument not specified.
- OS\_FLAG\_INVALID\_PGRP - null pointer passed instead of the event flag group handle.
- OS\_FLAG\_ERR\_NOT\_RDY - Flags not available.

### RETURN VALUE

The state of the flags in the event flag group.

### LIBRARY

OS\_FLAG.C (Prior to DC 8:UCOS2.LIB)

---

---

## OSFlagCreate

---

---

```
OS_FLAG_GRP * OSFlagCreate( OS_FLAGS flags, INT8U * err );
```

### DESCRIPTION

This function is called to create an event flag group.

### PARAMETERS

- |              |   |
|--------------|---|
| <b>flags</b> | Contains the initial value to store in the event flag group.  |
| <b>err</b>   | Pointer to an error code that will be returned to your application: <ul style="list-style-type: none"><li>• OS_NO_ERR - The call was successful.</li><li>• OS_ERR_CREATE_ISR - Attempt made to create an Event Flag from an ISR.</li><li>• OS_FLAG_GRP_DEPLETED - There are no more event flag groups</li></ul> |

### RETURN VALUE

A pointer to an event flag group or a null pointer if no more groups are available.

### LIBRARY

OS\_FLAG.C (Prior to DC 8:UCOS2.LIB)

---

---

## OSFlagDel

---

---

```
OS_FLAG_GRP * OSFlagDel( OS_FLAG_GRP * pgrp, INT8U opt, INT8U * err);
```

### DESCRIPTION

This function deletes an event flag group and readies all tasks pending on the event flag group. Note that:

- This function must be used with care. Tasks that would normally expect the presence of the event flag group must check the return code of `OSFlagAccept()` and `OSFlagPend()`.
- This call can potentially disable interrupts for a long time. The interrupt disable time is directly proportional to the number of tasks waiting on the event flag group.

### PARAMETERS

<b>pgrp</b>	Pointer to the desired event flag group.
<b>opt</b>	May be one of the following delete options: <ul style="list-style-type: none"><li>• <code>OS_DEL_NO_PEND</code> - Deletes the event flag group only if no task pending</li><li>• <code>OS_DEL_ALWAYS</code> - Deletes the event flag group even if tasks are waiting. In this case, all the tasks pending will be readied..</li></ul>
<b>err</b>	Pointer to an error code. May be one of the following values: <ul style="list-style-type: none"><li>• <code>OS_NO_ERR</code> - Success, the event flag group was deleted</li><li>• <code>OS_ERR_DEL_ISR</code> - If you attempted to delete the event flag group from an ISR</li><li>• <code>OS_FLAG_INVALID_PGRP</code> - If <code>pgrp</code> is a null pointer.</li><li>• <code>OS_ERR_EVENT_TYPE</code> - You are not pointing to an event flag group</li><li>• <code>OS_ERR_EVENT_TYPE</code> - If you didn't pass a pointer to an event flag group</li><li>• <code>OS_ERR_INVALID_OPT</code> - Invalid option was specified</li><li>• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the event flag group.</li></ul>

### RETURN VALUE

<b>pevent</b>	Error.
<code>(OS_EVENT *)0</code>	Semaphore was successfully deleted.

### LIBRARY

`OS_FLAG.C` (Prior to DC 8:UCOS2.LIB)

---

---

## OSFlagPend

---

---

```
OS_FLAGS OSFlagPend( OS_FLAG_GRP * pgrp, OS_FLAGS flags, INT8U
    wait_type, INT16U timeout, INT8U * err );
```

### DESCRIPTION

This function is called to wait for a combination of bits to be set in an event flag group. Your application can wait for ANY bit to be set or ALL bits to be set.

### PARAMETERS

<b>pgrp</b>	Pointer to the desired event flag group.
<b>flags</b>	Bit pattern indicating which bit(s) (i.e. flags) you wish to wait for. E.g. if your application wants to wait for bits 0 and 1 then <code>flags</code> should be 0x03.
<b>wait_type</b>	Specifies whether you want ALL bits to be set or ANY of the bits to be set. You can specify the following argument: <ul style="list-style-type: none"><li>• <code>OS_FLAG_WAIT_CLR_ALL</code> - You will wait for ALL bits in <code>mask</code> to be clear (0)</li><li>• <code>OS_FLAG_WAIT_SET_ALL</code> - You will wait for ALL bits in <code>mask</code> to be set (1)</li><li>• <code>OS_FLAG_WAIT_CLR_ANY</code> - You will wait for ANY bit in <code>mask</code> to be clear (0)</li><li>• <code>OS_FLAG_WAIT_SET_ANY</code> - You will wait for ANY bit in <code>mask</code> to be set (1)</li></ul> <p><b>Note:</b> Add <code>OS_FLAG_CONSUME</code> if you want the event flag to be consumed by the call. E.g., to wait for any flag in a group AND then clear the flags that are present, set the <code>wait_type</code> parameter to:</p> <pre>OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME</pre>
<b>timeout</b>	An optional timeout (in clock ticks) that your task will wait for the desired bit combination. If you specify 0, however, your task will wait forever at the specified event flag group or, until a message arrives.

---

---

## OSFlagPend (cont'd)

---

---

**err** Pointer to an error code. Possible values are:

- OS\_NO\_ERR - The desired bits have been set within the specified time-out.
- OS\_ERR\_PEND\_ISR - If you tried to PEND from an ISR.
- OS\_FLAG\_INVALID\_PGRP - If pgrp is a null pointer.
- OS\_ERR\_EVENT\_TYPE - You are not pointing to an event flag group
- OS\_TIMEOUT - The bit(s) have not been set in the specified time-out.
- OS\_FLAG\_ERR\_WAIT\_TYPE - You didn't specify a proper wait\_type argument.

### RETURN VALUE

The new state of the flags in the event flag group when the task is resumed or, 0 if a timeout or an error occurred.

### LIBRARY

OS\_FLAG.C (Prior to DC 8:UCOS2.LIB)

---

---

## OSFlagPost

---

---

```
OS_FLAGS OSFlagPost( OS_FLAG_GRP * pgrp, OS_FLAGS flags, INT8U opt,  
                    INT8U * err );
```

### DESCRIPTION

This function is called to set or clear some bits in an event flag group. The bits to set or clear are specified by a bitmask. Warnings:

- The execution time of this function depends on the number of tasks waiting on the event flag group.
- The amount of time interrupts are DISABLED depends on the number of tasks waiting on the event flag group.

### PARAMETERS

<b>pgrp</b>	Pointer to the desired event flag group.
<b>flags</b>	If <b>opt</b> (see below) is <code>OS_FLAG_SET</code> , each bit that is set in <b>flags</b> will set the corresponding bit in the event flag group. E.g., to set bits 0, 4 and 5 you would set <b>flags</b> to:  <code>0x31</code> (note, bit 0 is least significant bit)  If <b>opt</b> (see below) is <code>OS_FLAG_CLR</code> , each bit that is set in <b>flags</b> will CLEAR the corresponding bit in the event flag group. E.g., to clear bits 0, 4 and 5 you would specify <b>flags</b> as:  <code>0x31</code> (note, bit 0 is least significant bit)
<b>opt</b>	Indicates whether the flags will be:  set ( <code>OS_FLAG_SET</code> ), or cleared ( <code>OS_FLAG_CLR</code> )
<b>err</b>	Pointer to an error code. Valid values are: <ul style="list-style-type: none"><li>• <code>OS_NO_ERR</code> - The call was successful.</li><li>• <code>OS_FLAG_INVALID_PGRP</code> - null pointer passed.</li><li>• <code>OS_ERR_EVENT_TYPE</code> - Not pointing to an event flag group</li><li>• <code>OS_FLAG_INVALID_OPT</code> - Invalid option specified.</li></ul>

### RETURN VALUE

The new value of the event flags bits that are still set.

### LIBRARY

`OS_FLAG.C` (Prior to DC 8:UCOS2.LIB)

---

---

## OSFlagQuery

---

---

```
OS_FLAGS OSFlagQuery( OS_FLAG_GRP * pgrp, INT8U * err );
```

### DESCRIPTION

This function is used to check the value of the event flag group.

### PARAMETERS

<b>pgrp</b>	Pointer to the desired event flag group.
<b>err</b>	Pointer to an error code returned to the called: <ul style="list-style-type: none"><li>• OS_NO_ERR - The call was successful</li><li>• OS_FLAG_INVALID_PGRP - null pointer passed.</li><li>• OS_ERR_EVENT_TYPE - Not pointing to an event flag group</li></ul>

### RETURN VALUE

The current value of the event flag group.

### LIBRARY

OS\_FLAG.C (Prior to DC 8:UCOS2.LIB)



---

---

## OSInit

---

---

```
void OSInit( void );
```

### DESCRIPTION

Initializes  $\mu$ C/OS-II data; must be called before any other  $\mu$ C/OS-II functions are called.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTaskCreate, OSTaskCreateExt, OSStart

---

---

## OSMboxAccept

---

---

```
void * OSMboxAccept( OS_EVENT * pevent );
```

### DESCRIPTION

Checks the mailbox to see if a message is available. Unlike `OSMboxPend()`, `OSMboxAccept()` does not suspend the calling task if a message is not available.

### PARAMETERS

**pevent**            Pointer to the mailbox's event control block.

### RETURN VALUE

**!= (void \*)0**        This is the message in the mailbox if one is available. The mailbox is cleared so the next time `OSMboxAccept()` is called, the mailbox will be empty.

**== (void \*)0**        The mailbox is empty, or `pevent` is a null pointer, or you didn't pass the proper event pointer.

### LIBRARY

OS\_MBOX.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSMboxCreate, OSMboxPend, OSMboxPost, OSMboxQuery

---

---

## OSMboxCreate

---

---

```
OS_EVENT * OSMboxCreate( void * msg );
```

### DESCRIPTION

Creates a message mailbox if event control blocks are available.

### PARAMETERS

**msg**                    Pointer to a message to put in the mailbox. If this value is set to the null pointer (i.e., (void \*) 0) then the mailbox will be considered empty.

### RETURN VALUE

**!= (void \*) 0**            A pointer to the event control clock (OS\_EVENT) associated with the created mailbox.

**== (void \*) 0**            No event control blocks were available.

### LIBRARY

OS\_MBOX.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSMboxAccept, OSMboxPend, OSMboxPost, OSMboxQuery

---

---

## OSMboxDel

---

---

```
OS_EVENT * OSMboxDel( OS_EVENT * pevent, INT8U opt, INT8U * err );
```

### DESCRIPTION

This function deletes a mailbox and readies all tasks pending on the mailbox. Note that:

- This function must be used with care. Tasks that would normally expect the presence of the mailbox **MUST** check the return code of `OSMboxPend()`.
- `OSMboxAccept()` callers will not know that the intended mailbox has been deleted unless they check `pevent` to see that it's a null pointer.
- This call can potentially disable interrupts for a long time. The interrupt disable time is directly proportional to the number of tasks waiting on the mailbox.
- Because ALL tasks pending on the mailbox will be readied, you **MUST** be careful in applications where the mailbox is used for mutual exclusion because the resource(s) will no longer be guarded by the mailbox.

### PARAMETERS

<b>pevent</b>	Pointer to the event control block associated with the desired mailbox.
<b>opt</b>	May be one of the following delete options: <ul style="list-style-type: none"><li>• <code>OS_DEL_NO_PEND</code> - Delete mailbox only if no task pending</li><li>• <code>OS_DEL_ALWAYS</code> - Deletes the mailbox even if tasks are waiting. In this case, all the tasks pending will be readied.</li></ul>
<b>err</b>	Pointer to an error code that can contain one of the following values: <ul style="list-style-type: none"><li>• <code>OS_NO_ERR</code> - Call was successful; mailbox was deleted</li><li>• <code>OS_ERR_DEL_ISR</code> - Attempt to delete mailbox from ISR</li><li>• <code>OS_ERR_INVALID_OPT</code> - Invalid option was specified</li><li>• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the mailbox</li><li>• <code>OS_ERR_EVENT_TYPE</code> - No pointer passed to a mailbox</li><li>• <code>OS_ERR_PEVENT_NULL</code> - If <code>pevent</code> is a null pointer.</li></ul>

### RETURN VALUE

<code>!= (void *)0</code>	Is a pointer to the event control clock ( <code>OS_EVENT</code> ) associated with the created mailbox
<code>== (void *)0</code>	If no event control blocks were available

### LIBRARY

`OS_MBOX.C`

---

---

## OSMboxPend

---

---

```
void *OSMboxPend( OS_EVENT *pevent, INT16U timeout, INT8U *err );
```

### DESCRIPTION

Waits for a message to be sent to a mailbox.

### PARAMETERS

<b>pevent</b>	Pointer to mailbox's event control block.
<b>timeout</b>	Allows task to resume execution if a message was not received by the number of clock ticks specified. Specifying 0 means the task is willing to wait forever.
<b>err</b>	Pointer to a variable for holding an error code. Possible error messages are: <ul style="list-style-type: none"><li>• OS_NO_ERR: The call was successful and the task received a message.</li><li>• OS_TIMEOUT: A message was not received within the specified timeout</li><li>• OS_ERR_EVENT_TYPE: Invalid event type</li><li>• OS_ERR_PEND_ISR If this function was called from an ISR and the result would lead to a suspension.</li><li>• OS_ERR_PEVENT_NULL: If pevent is a null pointer</li></ul>

### RETURN VALUE

<b>!= (void *)0</b>	A pointer to the message received
<b>== (void *)0</b>	No message was received, or pevent is a null pointer, or the proper pointer to the event control block was not passed.

### LIBRARY

OS\_MBOX.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSMboxAccept, OSMboxCreate, OSMboxPost, OSMboxQuery

---

---

## OSMboxPost

---

---

```
INT8U OSMboxPost( OS_EVENT * pevent, void * msg );
```

### DESCRIPTION

Sends a message to the specified mailbox.

### PARAMETERS

<b>pevent</b>	Pointer to mailbox's event control block.
<b>msg</b>	Pointer to message to be posted. A null pointer must not be sent.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent.
<b>OS_MBOX_FULL</b>	The mailbox already contains a message. Only one message at a time can be sent and thus, the message <b>MUST</b> be consumed before another can be sent.
<b>OS_ERR_EVENT_TYPE</b>	Attempting to post to a non-mailbox.
<b>OS_ERR_PEVENT_NULL</b>	If <b>pevent</b> is a null pointer
<b>OS_ERR_POST_NULL_PTR</b>	If you are attempting to post a null pointer

### LIBRARY

OS\_MBOX.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSMboxAccept, OSMboxCreate, OSMboxPend, OSMboxQuery

---

---

## OSMboxPostOpt

---

---

```
INT8U OSMboxPostOpt( OS_EVENT * pevent, void * msg, INT8U opt );
```

### DESCRIPTION

This function sends a message to a mailbox.

**Note:** Interrupts can be disabled for a long time if you do a “broadcast.” The interrupt disable time is proportional to the number of tasks waiting on the mailbox.

### PARAMETERS

<b>pevent</b>	Pointer to mailbox’s event control block.
<b>msg</b>	Pointer to the message to send. A null pointer must not be sent.
<b>opt</b>	Determines the type of POST performed: <ul style="list-style-type: none"><li>• OS_POST_OPT_NONE - POST to a single waiting task (Identical to OS_MboxPost ())</li><li>• OS_POST_OPT_BROADCAST - POST to ALL tasks that are waiting on the mailbox</li></ul>

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent.
<b>OS_MBOX_FULL</b>	The mailbox already contains a message. Only one message at a time can be sent and thus, the message <b>MUST</b> be consumed before another can be sent.
<b>OS_ERR_EVENT_TYPE</b>	Attempting to post to a non-mailbox.
<b>OS_ERR_PEVENT_NULL</b>	If <b>pevent</b> is a null pointer
<b>OS_ERR_POST_NULL_PTR</b>	If you are attempting to post a null pointer

### LIBRARY

OS\_MBOX.C (Prior to DC 8:UCOS2.LIB)

---

---

## OSMboxQuery

---

---

```
INT8U OSMboxQuery( OS_EVENT * pevent, OS_MBOX_DATA * pdata );
```

### DESCRIPTION

Obtains information about a message mailbox.

### PARAMETERS

<b>pevent</b>	Pointer to message mailbox's event control block.
<b>pdata</b>	Pointer to a data structure for information about the message mailbox

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent.
------------------	---

<b>OS_ERR_EVENT_TYPE</b>	Attempting to obtain data from a non mailbox.
--------------------------	---

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSMboxAccept, OSMboxCreate, OSMboxPend, OSMboxPost

---

---

## OSMemCreate

---

---

```
OS_MEM * OSMemCreate( void * addr, INT32U nblks, INT32U blksize,  
    INT8U * err );
```

### DESCRIPTION

Creates a fixed-sized memory partition that will be managed by  $\mu$ C/OS-II.

### PARAMETERS

<b>addr</b>	Pointer to starting address of the partition.
<b>nblks</b>	Number of memory blocks to create in the partition.
<b>blksize</b>	The size (in bytes) of the memory blocks.
<b>err</b>	Pointer to variable containing an error message.

### RETURN VALUE

Pointer to the created memory partition control block if one is available, null pointer otherwise.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSMemGet, OSMemPut, OSMemQuery



---

---

## OSMemGet

---

---

```
void * OSMemGet( OS_MEM * pmem, INT8U * err );
```

### DESCRIPTION

Gets a memory block from the specified partition.

### PARAMETERS

<b>pmem</b>	Pointer to partition's memory control block
<b>err</b>	Pointer to variable containing an error message

### RETURN VALUE

Pointer to a memory block or a null pointer if an error condition is detected.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSMemCreate, OSMemPut, OSMemQuery

---

---

## OSMemPut

---

---

```
INT8U OSMemPut( OS_MEM * pmem, void * pblk );
```

### DESCRIPTION

Returns a memory block to a partition.

### PARAMETERS

<b>pmem</b>	Pointer to the partition's memory control block.
<b>pblk</b>	Pointer to the memory block being released.

### RETURN VALUE

<b>OS_NO_ERR</b>	The memory block was inserted into the partition.
<b>OS_MEM_FULL</b>	If returning a memory block to an already FULL memory partition. (More blocks were freed than allocated!)

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSMemCreate, OSMemGet, OSMemQuery

---

---

## OSMemQuery

---

---

```
INT8U OSMemQuery( OS_MEM * pmem, OS_MEM_DATA * pdata );
```

### DESCRIPTION

Determines the number of both free and used memory blocks in a memory partition.

### PARAMETERS

<code>pmem</code>	Pointer to partition's memory control block.
<code>pdata</code>	Pointer to structure for holding information about the partition.

### RETURN VALUE

<code>OS_NO_ERR</code>	This function always returns no error.
------------------------	--

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSMemCreate, OSMemGet, OSMemPut

---

---

## OSMutexAccept

---

---

```
INT8U OSMutexAccept( OS_EVENT * pevent, INT8U * err );
```

### DESCRIPTION

This function checks the mutual exclusion semaphore to see if a resource is available. Unlike `OSMutexPend()`, `OSMutexAccept()` does not suspend the calling task if the resource is not available or the event did not occur. This function cannot be called from an ISR because mutual exclusion semaphores are intended to be used by tasks only.

### PARAMETERS

**pevent**            Pointer to the event control block.

**err**                Pointer to an error code that will be returned to your application:

- `OS_NO_ERR` - if the call was successful.
- `OS_ERR_EVENT_TYPE` - if `pevent` is not a pointer to a mutex
- `OS_ERR_PEVENT_NULL` - `pevent` is a null pointer
- `OS_ERR_PEND_ISR` - if you called this function from an ISR

### RETURN VALUE

1: Success, the resource is available and the mutual exclusion semaphore is acquired.

0: Error, either the resource is not available, or you didn't pass a pointer to a mutual exclusion semaphore, or you called this function from an ISR.

### LIBRARY

`OS_MUTEX.C`

---

---

## OSMutexCreate

---

---

```
OS_EVENT *OSMutexCreate( INT8U prio, INT8U * err );
```

### DESCRIPTION

This function creates a mutual exclusion semaphore. Note that:

- The LEAST significant 8 bits of the OSEventCnt field of the mutex's event control block are used to hold the priority number of the task owning the mutex or 0xFF if no task owns the mutex.
- The MOST significant 8 bits of the OSEventCnt field of the mutex's event control block are used to hold the priority number to use to reduce priority inversion.

### PARAMETERS

<b>prio</b>	The priority to use when accessing the mutual exclusion semaphore. In other words, when the semaphore is acquired and a higher priority task attempts to obtain the semaphore then the priority of the task owning the semaphore is raised to this priority. It is assumed that you will specify a priority that is LOWER in value than ANY of the tasks competing for the mutex.
<b>err</b>	Pointer to error code that will be returned to your application: <ul style="list-style-type: none"><li>• OS_NO_ERR - if the call was successful.</li><li>• OS_ERR_CREATE_ISR - you attempted to create a mutex from an ISR</li><li>• OS_PRIO_EXIST - a task at the priority inheritance priority already exist.</li><li>• OS_ERR_PEVENT_NULL - no more event control blocks available.</li><li>• OS_PRIO_INVALID - if the priority you specify is higher than the maximum allowed (i.e. &gt; OS_LOWEST_PRIO)</li></ul>

### RETURN VALUE

<b>!= (void *)0</b>	Pointer to the event control clock (OS_EVENT) associated with the created mutex.
<b>== (void *)0</b>	Error detected.

### LIBRARY

OS\_MUTEX.C

---

---

## OSMutexDel

---

---

```
OS_EVENT *OSMutexDel( OS_EVENT * pevent, INT8U opt, INT8U * err );
```

### DESCRIPTION

This function deletes a mutual exclusion semaphore and readies all tasks pending on it. Note that:

- This function must be used with care. Tasks that would normally expect the presence of the mutex **MUST** check the return code of `OSMutexPend()`.
- This call can potentially disable interrupts for a long time. The interrupt disable time is directly proportional to the number of tasks waiting on the mutex.
- Because ALL tasks pending on the mutex will be readied, you **MUST** be careful because the resource(s) will no longer be guarded by the mutex.

### PARAMETERS

<b>pevent</b>	Pointer to mutex's event control block.
<b>opt</b>	May be one of the following delete options: <ul style="list-style-type: none"><li>• <code>OS_DEL_NO_PEND</code> - Delete mutex only if no task pending</li><li>• <code>OS_DEL_ALWAYS</code> - Deletes the mutex even if tasks are waiting. In this case, all pending tasks will be readied.</li></ul>
<b>err</b>	Pointer to an error code that can contain one of the following values: <ul style="list-style-type: none"><li>• <code>OS_NO_ERR</code> - The call was successful and the mutex was deleted</li><li>• <code>OS_ERR_DEL_ISR</code> - Attempted to delete the mutex from an ISR</li><li>• <code>OS_ERR_INVALID_OPT</code> - An invalid option was specified</li><li>• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the mutex</li><li>• <code>OS_ERR_EVENT_TYPE</code> - If you didn't pass a pointer to a mutex pointer.</li></ul>

### RETURN VALUE

<b>pevent</b>	On error.
<code>(OS_EVENT *) 0</code>	Mutex was deleted.

### LIBRARY

`OS_MUTEX.C`

---

---

## OSMutexPend

---

---

```
void OSMutexPend( OS_EVENT *pevent, INT16U timeout, INT8U *err );
```

### DESCRIPTION

This function waits for a mutual exclusion semaphore. Note that:

- The task that owns the Mutex **MUST NOT** pend on any other event while it owns the mutex.
- You **MUST NOT** change the priority of the task that owns the mutex.

### PARAMETERS

<b>pevent</b>	Pointer to mutex's event control block.
<b>timeout</b>	Optional timeout period (in clock ticks). If non-zero, your task will wait for the resource up to the amount of time specified by this argument. If you specify 0, however, your task will wait forever at the specified mutex or, until the resource becomes available.
<b>err</b>	Pointer to where an error message will be deposited. Possible error messages are: OS_NO_ERR - The call was successful and your task owns the mutex OS_TIMEOUT - The mutex was not available within the specified time. OS_ERR_EVENT_TYPE - If you didn't pass a pointer to a mutex OS_ERR_PEVENT_NULL - pevent is a null pointer OS_ERR_PEND_ISR - If you called this function from an ISR and the result would lead to a suspension.

### LIBRARY

OS\_MUTEX.C

---

---

## OSMutexPost

---

---

```
INT8U OSMutexPost( OS_EVENT * pevent );
```

### DESCRIPTION

This function signals a mutual exclusion semaphore.

### PARAMETERS

**pevent**                    Pointer to mutex's event control block.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the mutex was signaled.
<b>OS_ERR_EVENT_TYPE</b>	If you didn't pass a pointer to a mutex
<b>OS_ERR_PEVENT_NULL</b>	<b>pevent</b> is a null pointer
<b>OS_ERR_POST_ISR</b>	Attempted to post from an ISR (invalid for mutexes)
<b>OS_ERR_NOT_MUTEX_OWNER</b>	The task that did the post is NOT the owner of the MUTEX.

### LIBRARY

OS\_MUTEX.C



---

---

## OSMutexQuery

---

---

```
INT8U OSMutexQuery( OS_EVENT * pevent, OS_MUTEX_DATA * pdata );
```

### DESCRIPTION

This function obtains information about a mutex.

### PARAMETERS

<b>pevent</b>	Pointer to the event control block associated with the desired mutex.
<b>pdata</b>	Pointer to a structure that will contain information about the mutex.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent
<b>OS_ERR_QUERY_ISR</b>	Function was called from an ISR
<b>OS_ERR_PEVENT_NULL</b>	<b>pevent</b> is a null pointer
<b>OS_ERR_EVENT_TYPE</b>	Attempting to obtain data from a non mutex.

### LIBRARY

OS\_MUTEX.C

---

---

## OSQAccept

---

---

```
void * OSQAccept( OS_EVENT * pevent );
```

### DESCRIPTION

Checks the queue to see if a message is available. Unlike OSQPend(), with OSQAccept() the calling task is not suspended if a message is unavailable.

### PARAMETERS

**pevent**            Pointer to the message queue's event control block.

### RETURN VALUE

Pointer to message in the queue if one is available, null pointer otherwise.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSQCreate, OSQFlush, OSQPend, OSQPost, OSQPostFront, OSQQuery

---

---

## OSQCreate

---

---

```
OS_EVENT * OSQCreate( void ** start, INT16U qsize );
```

### DESCRIPTION

Creates a message queue if event control blocks are available.

### PARAMETERS

<b>start</b>	Pointer to the base address of the message queue storage area. The storage area <b>MUST</b> be declared an array of pointers to void: void *MessageStorage [qsize] .
<b>qsize</b>	Number of elements in the storage area.

### RETURN VALUE

Pointer to message queue's event control block or null pointer if no event control blocks were available.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSQAccept, OSQFlush, OSQPend, OSQPost, OSQPostFront, OSQQuery

---

---

## OSQDel

---

---

```
OS_EVENT * OSQDel( OS_EVENT * pevent, INT8U opt, INT8U * err );
```

### DESCRIPTION

Deletes a message queue and readies all tasks pending on the queue. Note that:

- This function must be used with care. Tasks that would normally expect the presence of the queue MUST check the return code of `OSQPend()`.
- `OSQAccept()` callers will not know that the intended queue has been deleted unless they check `pevent` to see that it's a null pointer.
- This call can potentially disable interrupts for a long time. The interrupt disable time is directly proportional to the number of tasks waiting on the queue.
- Because all tasks pending on the queue will be readied, you must be careful in applications where the queue is used for mutual exclusion because the resource(s) will no longer be guarded by the queue.
- If the storage for the message queue was allocated dynamically (i.e., using a `malloc()` type call) then your application must release the memory storage by call the counterpart call of the dynamic allocation scheme used. If the queue storage was created statically then, the storage can be reused.

### PARAMETERS

<b>pevent</b>	Pointer to the queue's event control block.
<b>opt</b>	May be one of the following delete options: <ul style="list-style-type: none"><li>• <code>OS_DEL_NO_PEND</code> - Delete queue only if no task pending</li><li>• <code>OS_DEL_ALWAYS</code> - Deletes the queue even if tasks are waiting. In this case, all the tasks pending will be readied.</li></ul>
<b>err</b>	Pointer to an error code that can contain one of the following: <ul style="list-style-type: none"><li>• <code>OS_NO_ERR</code> - Call was successful and queue was deleted</li><li>• <code>OS_ERR_DEL_ISR</code> - Attempt to delete queue from an ISR</li><li>• <code>OS_ERR_INVALID_OPT</code> - Invalid option was specified</li><li>• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the queue</li><li>• <code>OS_ERR_EVENT_TYPE</code> - You didn't pass a pointer to a queue</li><li>• <code>OS_ERR_PEVENT_NULL</code> - If <code>pevent</code> is a null pointer.</li></ul>

### RETURN VALUE

<b>pevent</b>	Error
<code>(OS_EVENT *)0</code>	The queue was successfully deleted.

### LIBRARY

`OS_Q.C` (Prior to DC 8:UCOS2.LIB)

---

---

## OSQFlush

---

---

```
INT8U OSQFlush( OS_EVENT * pevent );
```

### DESCRIPTION

Flushes the contents of the message queue.

### PARAMETERS

**pevent**                    Pointer to message queue's event control block.

### RETURN VALUE

**OS\_NO\_ERR**                    Success.  
**OS\_ERR\_EVENT\_TYPE**        A pointer to a queue was not passed.  
**OS\_ERR\_PEVENT\_NULL**      If *pevent* is a null pointer.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSQAccept, OSQCreate, OSQPend, OSQPost, OSQPostFront, OSQQuery

---

---

## OSQPend

---

---

```
void * OSQPend( OS_EVENT * pevent, INT16U timeout, INT8U * err );
```

### DESCRIPTION

Waits for a message to be sent to a queue.

### PARAMETERS

<b>pevent</b>	Pointer to message queue's event control block.
<b>timeout</b>	Allow task to resume execution if a message was not received by the number of clock ticks specified. Specifying 0 means the task is willing to wait forever.
<b>err</b>	Pointer to a variable for holding an error code.

### RETURN VALUE

Pointer to a message or, if a timeout occurs, a null pointer.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPost, OSQPostFront, OSQQuery

---

---

## OSQPost

---

---

```
INT8U OSQPost( OS_EVENT * pevent, void * msg );
```

### DESCRIPTION

Sends a message to the specified queue.

### PARAMETERS

<b>pevent</b>	Pointer to message queue's event control block.
<b>msg</b>	Pointer to the message to send. A null pointer must not be sent.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent.
<b>OS_Q_FULL</b>	The queue cannot accept any more messages because it is full.
<b>OS_ERR_EVENT_TYPE</b>	If a pointer to a queue not passed.
<b>OS_ERR_PEVENT_NULL</b>	If <b>pevent</b> is a null pointer.
<b>OS_ERR_POST_NULL_PTR</b>	If attempting to post to a null pointer.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPend, OSQPostFront, OSQQuery

---

---

## OSQPostFront

---

---

```
INT8U OSQPostFront( OS_EVENT * pevent, void * msg );
```

### DESCRIPTION

Sends a message to the specified queue, but unlike `OSQPost()`, the message is posted at the front instead of the end of the queue. Using `OSQPostFront()` allows 'priority' messages to be sent.

### PARAMETERS

**pevent**            Pointer to message queue's event control block.

**msg**                Pointer to the message to send. A null pointer must not be sent.

### RETURN VALUE

**OS\_NO\_ERR**            The call was successful and the message was sent.

**OS\_Q\_FULL**            The queue cannot accept any more messages because it is full.

**OS\_ERR\_EVENT\_TYPE**    A pointer to a queue was not passed.

**OS\_ERR\_PEVENT\_NULL**   If `pevent` is a null pointer.

**OS\_ERR\_POST\_NULL\_PTR** Attempting to post to a non mailbox.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

`OSQAccept`, `OSQCreate`, `OSQFlush`, `OSQPend`, `OSQPost`, `OSQQuery`



---

---

## OSQPostOpt

---

---

```
INT8U OSQPostOpt( OS_EVENT * pevent, void * msg, INT8U opt );
```

### DESCRIPTION

This function sends a message to a queue. This call has been added to reduce code size since it can replace both `OSQPost()` and `OSQPostFront()`. Also, this function adds the capability to broadcast a message to all tasks waiting on the message queue.

**Note:** Interrupts can be disabled for a long time if you do a “broadcast.” In fact, the interrupt disable time is proportional to the number of tasks waiting on the queue.

### PARAMETERS

<b>pevent</b>	Pointer to message queue’s event control block.
<b>msg</b>	Pointer to the message to send. A null pointer must not be sent.
<b>opt</b>	Determines the type of POST performed: <ul style="list-style-type: none"><li>• <code>OS_POST_OPT_NONE</code> - POST to a single waiting task (Identical to <code>OSQPost()</code>)</li><li>• <code>OS_POST_OPT_BROADCAST</code> - POST to ALL tasks that are waiting on the queue</li><li>• <code>OS_POST_OPT_FRONT</code> - POST as LIFO (Simulates <code>OSQPostFront()</code>)</li></ul> The last 2 flags may be combined: <ul style="list-style-type: none"><li>• <code>OS_POST_OPT_FRONT + OS_POST_OPT_BROADCAST</code> - is identical to <code>OSQPostFront()</code> except that it will broadcast <code>msg</code> to all waiting tasks.</li></ul>

### RETURN VALUE

<code>OS_NO_ERR</code>	The call was successful and the message was sent.
<code>OS_Q_FULL</code>	The queue is full, cannot accept any more messages.
<code>OS_ERR_EVENT_TYPE</code>	A pointer to a queue was not passed.
<code>OS_ERR_PEVENT_NULL</code>	If <code>pevent</code> is a null pointer.
<code>OS_ERR_POST_NULL_PTR</code>	Attempting to post a null pointer.

### LIBRARY

`OS_Q.C` (Prior to DC 8:UCOS2.LIB)

---

---

## OSQQuery

---

---

```
INT8U OSQQuery( OS_EVENT * pevent, OS_Q_DATA * pdata );
```

### DESCRIPTION

Obtains information about a message queue.

### PARAMETERS

<b>pevent</b>	Pointer to message queue's event control block.
<b>pdata</b>	Pointer to a data structure for message queue information.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent
<b>OS_ERR_EVENT_TYPE</b>	Attempting to obtain data from a non queue.
<b>OS_ERR_PEVENT_NULL</b>	If pevent is a null pointer.

### LIBRARY

OS\_Q.C (Prior to DC 8:UCOS2.LIB)

### SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPend, OSQPost, OSQPostFront

---

---

## OSSchedLock

---

---

```
void OSSchedLock( void );
```

### DESCRIPTION

Prevents task rescheduling. This allows an application to prevent context switches until it is ready for them. There must be a matched call to `OSSchedUnlock()` for every call to `OSSchedLock()`.

### LIBRARY

UCOS2.LIB

### SEE ALSO

`OSSchedUnlock`

---

---

## OSSchedUnlock

---

---

```
void OSSchedUnlock( void );
```

### DESCRIPTION

Allow task rescheduling. There must be a matched call to `OSSchedUnlock()` for every call to `OSSchedLock()`.

### LIBRARY

UCOS2.LIB

### SEE ALSO

`OSSchedLock`

---

---

## OSSemAccept

---

---

```
INT16U OSSemAccept( OS_EVENT * pevent );
```

### DESCRIPTION

This function checks the semaphore to see if a resource is available or if an event occurred. Unlike `OSSemPend()`, `OSSemAccept()` does not suspend the calling task if the resource is not available or the event did not occur.

### PARAMETERS

**pevent**            Pointer to the desired semaphore's event control block

### RETURN VALUE

Semaphore value:

If >0, semaphore value is decremented; value is returned before the decrement.

If 0, then either resource is unavailable, event did not occur, or null or invalid pointer was passed to the function.

### LIBRARY

UCOS2.LIB

### SEE ALSO

`OSSemCreate`, `OSSemPend`, `OSSemPost`, `OSSemQuery`

---

---

## OSSemCreate

---

---

```
OS_EVENT * OSemCreate( INT16U cnt );
```

### DESCRIPTION

Creates a semaphore.

### PARAMETERS

**cnt**                    The initial value of the semaphore.

### RETURN VALUE

Pointer to the event control block (OS\_EVENT) associated with the created semaphore, or null if no event control block is available.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSSemAccept, OSemPend, OSemPost, OSemQuery

---

---

## OSemPend

---

---

```
void OSemPend( OS_EVENT * pevent, INT16U timeout, INT8U * err );
```

### DESCRIPTION

Waits on a semaphore.

### PARAMETERS

**pevent**                Pointer to the desired semaphore's event control block

**timeout**              Time in clock ticks to wait for the resource. If 0, the task will wait until the resource becomes available or the event occurs.

**err**                    Pointer to error message.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSSemAccept, OSemCreate, OSemPost, OSemQuery

---

---

## OSSemPost

---

---

```
INT8U OSSemPost ( OS_EVENT * pevent );
```

### DESCRIPTION

This function signals a semaphore.

### PARAMETERS

**pevent**                    Pointer to the desired semaphore's event control block

### RETURN VALUE

**OS\_NO\_ERR**                    The call was successful and the semaphore was signaled.

**OS\_SEM\_OVF**                    If the semaphore count exceeded its limit. In other words, you have signalled the semaphore more often than you waited on it with either `OSSemAccept()` or `OSSemPend()`.

**OS\_ERR\_EVENT\_TYPE**            If a pointer to a semaphore not passed.

**OS\_ERR\_PEVENT\_NULL**            If `pevent` is a null pointer.

### LIBRARY

UCOS2.LIB

### SEE ALSO

`OSSemAccept`, `OSSemCreate`, `OSSemPend`, `OSSemQuery`

---

---

## OSSemQuery

---

---

```
INT8U OSSemQuery( OS_EVENT * pevent, OS_SEM_DATA * pdata );
```

### DESCRIPTION

Obtains information about a semaphore.

### PARAMETERS

<b>pevent</b>	Pointer to the desired semaphore's event control block
<b>pdata</b>	Pointer to a data structure that will hold information about the semaphore.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful and the message was sent.
<b>OS_ERR_EVENT_TYPE</b>	Attempting to obtain data from a non semaphore.
<b>OS_ERR_PEVENT_NULL</b>	If the <code>pevent</code> parameter is a null pointer.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSSemAccept, OSSemCreate, OSSemPend, OSSemPost

---

---

## OSSetTickPerSec

---

---

```
INT16U OSetTickPerSec( INT16U TicksPerSec );
```

### DESCRIPTION

Sets the amount of ticks per second (from 1 - 2048). Ticks per second defaults to 64. If this function is used, the `#define OS_TICKS_PER_SEC` needs to be changed so that the time delay functions work correctly. Since this function uses integer division, the actual ticks per second may be slightly different than the desired ticks per second.

### PARAMETERS

**TicksPerSec** Unsigned 16-bit integer.

### RETURN VALUE

The actual ticks per second set, as an unsigned 16-bit integer.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSStart

---

---

## OSStart

---

---

```
void OSStart( void );
```

### DESCRIPTION

Starts the multitasking process, allowing  $\mu$ C/OS-II to manage the tasks that have been created. Before `OSStart()` is called, `OSInit()` MUST have been called and at least one task MUST have been created. This function calls `OSStartHighRdy` which calls `OSTaskSwHook` and sets `OSRunning` to TRUE.

### LIBRARY

UCOS2.LIB

### SEE ALSO

`OSTaskCreate`, `OSTaskCreateExt`



---

---

## OSStatInit

---

---

```
void OSStatInit( void );
```

### DESCRIPTION

Determines CPU usage.

### LIBRARY

UCOS2.LIB

---

---

## OSTaskChangePrio

---

---

```
INT8U OSTaskChangePrio( INT8U oldprio, INT8U newprio );
```

### DESCRIPTION

Allows a task's priority to be changed dynamically. Note that the new priority **MUST** be available.

### PARAMETERS

<b>oldprio</b>	The priority level to change from.
<b>newprio</b>	The priority level to change to.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful.
<b>OS_PRIO_INVALID</b>	The priority specified is higher than the maximum allowed (i.e. $\geq$ OS_LOWEST_PRIO).
<b>OS_PRIO_EXIST</b>	The new priority already exist
<b>OS_PRIO_ERR</b>	There is no task with the specified OLD priority (i.e. the OLD task does not exist).

### LIBRARY

UCOS2.LIB

---

---

## OSTaskCreate

---

---

```
INT8U OSTaskCreate( void (*task)(), void *pdata, INT16U stk_size,
    INT8U prio );
```

### DESCRIPTION

Creates a task to be managed by  $\mu$ C/OS-II. Tasks can either be created prior to the start of multitasking or by a running task. A task cannot be created by an ISR.

### PARAMETERS

<b>task</b>	Pointer to the task's starting address.
<b>pdata</b>	Pointer to a task's initial parameters.
<b>stk_size</b>	Number of bytes of the stack.
<b>prior</b>	The task's unique priority number.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful.
<b>OS_PRIO_EXIT</b>	Task priority already exists (each task MUST have a unique priority).
<b>OS_PRIO_INVALID</b>	The priority specified is higher than the maximum allowed (i.e. $\geq$ OS_LOWEST_PRIO).

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTaskCreateExt

---

---

## OSTaskCreateExt

---

---

```
INT8U OSTaskCreateExt( void (* task)(), void * pdata, INT8U prio,
    INT16U id, INT16U stk_size, void * pext, INT16U opt );
```

### DESCRIPTION

Creates a task to be managed by  $\mu$ C/OS-II. Tasks can either be created prior to the start of multitasking or by a running task. A task cannot be created by an ISR. This function is similar to `OSTaskCreate()` except that it allows additional information about a task to be specified.

### PARAMETERS

<b>task</b>	Pointer to task's code.
<b>pdata</b>	Pointer to optional data area; used to pass parameters to the task at start of execution.
<b>prio</b>	The task's unique priority number; the lower the number the higher the priority.
<b>id</b>	The task's identification number (0...65535).
<b>stk_size</b>	Size of the stack in number of elements. If <code>OS_STK</code> is set to <code>INT8U</code> , <code>stk_size</code> corresponds to the number of bytes available. If <code>OS_STK</code> is set to <code>INT16U</code> , <code>stk_size</code> contains the number of 16-bit entries available. Finally, if <code>OS_STK</code> is set to <code>INT32U</code> , <code>stk_size</code> contains the number of 32-bit entries available on the stack.
<b>pext</b>	Pointer to a user-supplied Task Control Block (TCB) extension.
<b>opt</b>	The lower 8 bits are reserved by $\mu$ C/OS-II. The upper 8 bits control application-specific options. Select an option by setting the corresponding bit(s).

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful.
<b>OS_Prio_EXIT</b>	Task priority already exists (each task MUST have a unique priority).
<b>OS_Prio_INVALID</b>	The priority specified is higher than the maximum allowed (i.e. $\geq$ <code>OS_LOWEST_Prio</code> ).

### LIBRARY

`UCOS2.LIB`

### SEE ALSO

`OSTaskCreate`

---

---

## OSTaskCreateHook

---

---

```
void OSTaskCreateHook( OS_TCB * ptcb );
```

### DESCRIPTION

Called by  $\mu$ C/OS-II whenever a task is created. This call-back function resides in `UCOS2.LIB` and extends functionality during task creation by allowing additional information to be passed to the kernel, anything associated with a task. This function can also be used to trigger other hardware, such as an oscilloscope. Interrupts are disabled during this call, therefore, it is recommended that code be kept to a minimum.

### PARAMETERS

**ptcb**                    Pointer to the TCB of the task being created.

### LIBRARY

`UCOS2.LIB`

### SEE ALSO

`OSTaskDelHook`

---

---

## OSTaskDel

---

---

```
INT8U OSTaskDel( INT8U prio );
```

### DESCRIPTION

Deletes a task. The calling task can delete itself by passing either its own priority number or `OS_PRIO_SELF` if it doesn't know its priority number. The deleted task is returned to the dormant state and can be re-activated by creating the deleted task again.

### PARAMETERS

`prio`                    Task's priority number.

### RETURN VALUE

<code>OS_NO_ERR</code>	The call was successful.
<code>OS_TASK_DEL_IDLE</code>	Attempting to delete $\mu$ C/OS-II's idle task.
<code>OS_PRIO_INVALID</code>	The priority specified is higher than the maximum allowed (i.e. $\geq$ <code>OS_LOWEST_PRIO</code> ) or, <code>OS_PRIO_SELF</code> not specified.
<code>OS_TASK_DEL_ERR</code>	The task to delete does not exist.
<code>OS_TASK_DEL_ISR</code>	Attempting to delete a task from an ISR.

### LIBRARY

`UCOS2.LIB`

### SEE ALSO

`OSTaskDelReq`

---

---

## OSTaskDelHook

---

---

```
void OSTaskDelHook( OS_TCB * ptcb );
```

### DESCRIPTION

Called by  $\mu$ C/OS-II whenever a task is deleted. This call-back function resides in UCOS2.LIB. Interrupts are disabled during this call, therefore, it is recommended that code be kept to a minimum.

### PARAMETERS

**ptcb**                    Pointer to TCB of task being deleted.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTaskCreateHook

---

---

## OSTaskDelReq

---

---

```
INT8U OSTaskDelReq( INT8U prio );
```

### DESCRIPTION

Notifies a task to delete itself. A well-behaved task is deleted when it regains control of the CPU by calling `OSTaskDelReq (OSTaskDelReq)` and monitoring the return value.

### PARAMETERS

**prio**                    The priority of the task that is being asked to delete itself. `OS_PRIO_SELF` is used when asking whether another task wants the current task to be deleted.

### RETURN VALUE

<code>OS_NO_ERR</code>	The task exists and the request has been registered.
<code>OS_TASK_NOT_EXIST</code>	The task has been deleted. This allows the caller to know whether the request has been executed.
<code>OS_TASK_DEL_IDLE</code>	If requesting to delete uC/OS-II's idletask.
<code>OS_PRIO_INVALID</code>	The priority specified is higher than the maximum allowed (i.e. $\geq$ <code>OS_LOWEST_PRIO</code> ) or, <code>OS_PRIO_SELF</code> is not specified.
<code>OS_TASK_DEL_REQ</code>	A task (possibly another task) requested that the running task be deleted.

### LIBRARY

`UCOS2.LIB`

### SEE ALSO

`OSTaskDel`

---

---

## OSTaskIdleHook

---

---

```
void OSTaskIdleHook( void );
```

### DESCRIPTION

This function is called by the idle task. This hook has been added to allow you to do such things as STOP the CPU to conserve power. Interrupts are enabled during this call.

### LIBRARY

UCOS2.LIB

---

---

## OSTaskQuery

---

---

```
INT8U OSTaskQuery( INT8U prio, OS_TCB * pdata );
```

### DESCRIPTION

Obtains a copy of the requested task's task control block (TCB).

### PARAMETERS

<b>prio</b>	Priority number of the task.
<b>pdata</b>	Pointer to task's TCB.

### RETURN VALUE

<b>OS_NO_ERR</b>	The requested task is suspended.
<b>OS_PRIO_INVALID</b>	The priority you specify is higher than the maximum allowed (i.e. $\geq$ OS_LOWEST_PRIO) or, OS_PRIO_SELF is not specified.
<b>OS_PRIO_ERR</b>	The desired task has not been created.

### LIBRARY

UCOS2.LIB



---

---

## OSTaskResume

---

---

```
INT8U OSTaskResume( INT8U prio );
```

### DESCRIPTION

Resumes a suspended task. This is the only call that will remove an explicit task suspension.

### PARAMETERS

**prio**                    The priority of the task to resume.

### RETURN VALUE

**OS\_NO\_ERR**                The requested task is resumed.

**OS\_PRIO\_INVALID**        The priority specified is higher than the maximum allowed (i.e.  $\geq$  OS\_LOWEST\_PRIO).

**OS\_TASK\_NOT\_SUSPENDED** The task to resume has not been suspended.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTaskSuspend

---

---

## OSTaskStatHook

---

---

```
void OSTaskStatHook( void );
```

### DESCRIPTION

Called every second by  $\mu$ C/OS-II's statistics task. This function resides in UCOS2.LIB and allows an application to add functionality to the statistics task.

### LIBRARY

UCOS2.LIB

---

---

## OSTaskStkChk

---

---

```
INT8U OSTaskStkChk( INT8U prio, OS_STK_DATA * pdata );
```

### DESCRIPTION

Check the amount of free memory on the stack of the specified task.

### PARAMETERS

<b>prio</b>	The task's priority.
<b>pdata</b>	Pointer to a data structure of type OS_STK_DATA.

### RETURN VALUE

<b>OS_NO_ERR</b>	The call was successful.
<b>OS_PRIO_INVALID</b>	The priority you specify is higher than the maximum allowed (i.e. > OS_LOWEST_PRIO) or, OS_PRIO_SELF not specified.
<b>OS_TASK_NOT_EXIST</b>	The desired task has not been created.
<b>OS_TASK_OPT_ERR</b>	If OS_TASK_OPT_STK_CHK was NOT specified when the task was created.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTaskCreateExt

---

---

## OSTaskSuspend

---

---

```
INT8U OSTaskSuspend( INT8U prio );
```

### DESCRIPTION

Suspends a task. The task can be the calling task if the priority passed to `OSTaskSuspend()` is the priority of the calling task or `OS_PRIO_SELF`. This function should be used with great care. If a task is suspended that is waiting for an event (i.e., a message, a semaphore, a queue...) the task will be prevented from running when the event arrives.

### PARAMETERS

**prio**                    The priority of the task to suspend.

### RETURN VALUE

<code>OS_NO_ERR</code>	The requested task is suspended.
<code>OS_TASK_SUS_IDLE</code>	Attempting to suspend the idle task (not allowed).
<code>OS_PRIO_INVALID</code>	The priority specified is higher than the maximum allowed (i.e. $\geq$ <code>OS_LOWEST_PRIO</code> ) or, <code>OS_PRIO_SELF</code> is not specified.
<code>OS_TASK_SUS_PRIO</code>	The task to suspend does not exist.

### LIBRARY

`UCOS2.LIB`

### SEE ALSO

`OSTaskResume`

---

---

## OSTaskSwHook

---

---

```
void OSTaskSwHook( void );
```

### DESCRIPTION

Called whenever a context switch happens. The task control block (TCB) for the task that is ready to run is accessed via the global variable OSTCBHighRdy, and the TCB for the task that is being switched out is accessed via the global variable OSTCBCur.

### LIBRARY

UCOS2.LIB

---

---

## OSTCBInitHook

---

---

```
void OSTCBInitHook( OS_TCB * ptcb );
```

### DESCRIPTION

This function is called by OSTCBInit () after setting up most of the task control block (TCB). Interrupts may or may not be enabled during this call.

### PARAMETER

**ptcb**                    Pointer to the TCB of the task being created.

### LIBRARY

UCOS2.LIB

---

---

## OSTimeDly

---

---

```
void OSTimeDly( INT16U ticks );
```

### DESCRIPTION

Delays execution of the task for the specified number of clock ticks. No delay will result if `ticks` is 0. If `ticks` is >0, then a context switch will result.

### PARAMETERS

**ticks**                      Number of clock ticks to delay the task.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTimeDlyHMSM, OSTimeDlyResume, OSTimeDlySec

---

---

## OSTimeDlyHMSM

---

---

```
INT8U OSTimeDlyHMSM( INT8U hours, INT8U minutes, INT8U seconds,  
                    INT16U milli );
```

### DESCRIPTION

Delays execution of the task until specified amount of time expires. This call allows the delay to be specified in hours, minutes, seconds and milliseconds instead of ticks. The resolution on the milliseconds depends on the tick rate. For example, a 10 ms delay is not possible if the ticker interrupts every 100 ms. In this case, the delay would be set to 0. The actual delay is rounded to the nearest tick.

### PARAMETERS

<b>hours</b>	Number of hours that the task will be delayed (max. is 255)
<b>minutes</b>	Number of minutes (max. 59)
<b>seconds</b>	Number of seconds (max. 59)
<b>milli</b>	Number of milliseconds (max. 999)

### RETURN VALUE

<b>OS_NO_ERR</b>	Execution delay of task was successful
<b>OS_TIME_INVALID_MINUTES</b>	Minutes parameter out of range
<b>OS_TIME_INVALID_SECONDS</b>	Seconds parameter out of range
<b>OS_TIME_INVALID_MS</b>	Milliseconds parameter out of range
<b>OS_TIME_ZERO_DLY</b>	

### LIBRARY

OS\_TIME.C (Prior to DC 8:ucos2.lib)

### SEE ALSO

OSTimeDly, OSTimeDlyResume, OSTimeDlySec

---

---

## OSTimeDlyResume

---

---

```
INT8U OSTimeDlyResume( INT8U prio );
```

### DESCRIPTION

Resumes a task that has been delayed through a call to either `OSTimeDly()` or `OSTimeDlyHMSM()`. Note that this function **MUST NOT** be called to resume a task that is waiting for an event with timeout. This situation would make the task look like a timeout occurred (unless this is the desired effect). Also, a task cannot be resumed that has called `OSTimeDlyHMSM()` with a combined time that exceeds 65535 clock ticks. In other words, if the clock tick runs at 100 Hz then, a delayed task will not be able to be resumed that called `OSTimeDlyHMSM(0, 10, 55, 350)` or higher.

### PARAMETERS

**prio**                      Priority of the task to resume.

### RETURN VALUE

<code>OS_NO_ERR</code>	Task has been resumed.
<code>OS_PRIO_INVALID</code>	The priority you specify is higher than the maximum allowed (i.e. $\geq$ <code>OS_LOWEST_PRIO</code> ).
<code>OS_TIME_NOT_DLY</code>	Task is not waiting for time to expire.
<code>OS_TASK_NOT_EXIST</code>	The desired task has not been created.

### LIBRARY

`UCOS2.LIB`

### SEE ALSO

`OSTimeDly`, `OSTimeDlyHMSM`, `OSTimeDlySec`

---

---

## OSTimeDlySec

---

---

```
INT8U OSTimeDlySec( INT16U seconds );
```

### DESCRIPTION

Delays execution of the task until `seconds` expires. This is a low-overhead version of `OSTimeDlyHMSM` for seconds only.

### PARAMETERS

**seconds**            The number of seconds to delay.

### RETURN VALUE

**OS\_NO\_ERR**            The call was successful.  
**OS\_TIME\_ZERO\_DLY**    A delay of zero seconds was requested.

### LIBRARY

UCOS2.LIB

### SEE ALSO

`OSTimeDly`, `OSTimeDlyHMSM`, `OSTimeDlyResume`



---

---

## OSTimeGet

---

---

```
INT32U OSTimeGet( void );
```

### DESCRIPTION

Obtain the current value of the 32-bit counter that keeps track of the number of clock ticks.

### RETURN VALUE

The current value of OSTime.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTimeSet

---

---

## OSTimeSet

---

---

```
void OSTimeSet( INT32U ticks );
```

### DESCRIPTION

Sets the 32-bit counter that keeps track of the number of clock ticks.

### PARAMETERS

**ticks**            The value to set OSTime to.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTimeGet

---

---

## OSTimeTick

---

---

```
void OSTimeTick( void );
```

### DESCRIPTION

This function takes care of the processing necessary at the occurrence of each system tick. This function is called from the BIOS timer interrupt ISR, but can also be called from a high priority task. The user definable `OSTimeTickHook()` is called from this function and allows for extra application specific processing to be performed at each tick. Since `OSTimeTickHook()` is called during an interrupt, it should perform minimal processing as it will directly affect interrupt latency.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTimeTickHook

---

---

## OSTimeTickHook

---

---

```
void OSTimeTickHook( void );
```

### DESCRIPTION

This function, as included with Dynamic C, is a stub that does nothing except return. It is called every clock tick. Code in this function should be kept to a minimum as it will directly affect interrupt latency. This function must preserve any registers it uses other than the ones that are preserved at the beginning of the periodic interrupt (`periodic_isr` in `VDRIVER.LIB`), and therefore should be written in assembly. At the time of this writing, the registers saved by `periodic_isr` are: AF,IP,HL,DE and IX.

### LIBRARY

UCOS2.LIB

### SEE ALSO

OSTimeTick

---

---

## OSVersion

---

---

```
INT16U OSVersion( void );
```

### DESCRIPTION

Returns the version number of  $\mu$ C/OS-II. The returned value corresponds to  $\mu$ C/OS-II's version number multiplied by 100; i.e., version 2.00 would be returned as 200.

### RETURN VALUE

Version number multiplied by 100.

### LIBRARY

UCOS2.LIB

---

---

## outchrs

---

---

```
char outchrs( char c, int n, int (*putc) () );
```

### DESCRIPTION

Use `putc` to output `n` times the character `c`.

### PARAMETERS

<code>c</code>	Character to output
<code>n</code>	Number of times to output
<code>putc</code>	Routine to output one character. The function pointed to by <code>putc</code> should take a character argument.

### RETURN VALUE

The character in parameter `c`.

### LIBRARY

STDIO.LIB

### SEE ALSO

`outstr`

---

---

## outstr

---

---

```
char * outstr( char * string, int (*putc)() );
```

### DESCRIPTION

Output the string pointed to by `string` via calls to `putc`. `putc` should take a one-character parameter.

### PARAMETERS

<code>string</code>	String to output
<code>putc</code>	Routine to output one character. The function pointed to by <code>putc</code> should take a character argument.

### RETURN VALUE

Pointer to null at end of string.

### LIBRARY

STDIO.LIB

### SEE ALSO

`outchrs`

---

---

## paddr

---

---

```
unsigned long paddr( void * pointer );
```

### DESCRIPTION

Converts a logical pointer into its physical address. This function is compatible with both shared and separate I&D space compile modes. Use caution when converting a pointer in the xmem window, i.e., in the range 0xE000 to 0xFFFF, as this function will return the physical address based on the XPC on entry.

### PARAMETERS

**pointer**            The pointer to convert.

### RETURN VALUE

The physical address of the pointer.

### LIBRARY

XMEM.LIB

### SEE ALSO

paddrDS, paddrSS

---

---

## paddrDS

---

---

```
unsigned long paddrDS( void * pointer );
```

### DESCRIPTION

Converts a "Data Segment" logical pointer into its physical address. This function assumes the pointer points to static (excluding bbram) data, which eliminates some runtime testing as compared with the more general function, `paddr ( )`.

`paddrDS ( )` will generate incorrect results if used for:

- addresses in the root code (that is, program code or constants)
- bbram (only available in fast RAM compile mode)
- stack (that is, auto variables)
- xmem segments

### PARAMETERS

**pointer**            Logical static (non-bbram) data pointer to convert.

### RETURN VALUE

The physical address of the pointer.

### LIBRARY

XMEM.LIB

### SEE ALSO

`paddr`, `paddrSS`

---

---

## paddrSS

---

---

```
unsigned long paddrSS( void * pointer );
```

### DESCRIPTION

Convert a logical pointer into its physical address. This function assumes the pointer points to data in the stack segment, which eliminates some runtime testing compared with the more general function, `paddr()`. The stack segment is used to store `auto` data items. This function will generate incorrect results if used for addresses in the root code (i.e. program code or constants), data (i.e. statically allocated variables), or `xmem` segments.

### PARAMETERS

**pointer**            The pointer to convert, pointing to stack (auto) data.

### RETURN VALUE

The physical address of the pointer.

### LIBRARY

`XMEM.LIB`

### SEE ALSO

`paddr`, `paddrDS`

---

---

## **palloc**

---

---

```
void * palloc( Pool_t * p );
```

### **DESCRIPTION**

Return next available free element from the given pool. Eventually, your application should return this element to the pool using `pfree()` to avoid memory leaks.

Assembler code can call `palloc_fast()` instead.

### **PARAMETERS**

**p** Pool handle structure, as previously passed to `pool_init()`.

### **RETURN VALUE**

Null: No free elements available

Otherwise, pointer to an element

### **LIBRARY**

POOL.LIB

### **SEE ALSO**

`pool_init`, `palloc`, `pfree`, `phwm`, `pavail`, `palloc_fast`, `pxalloc`,  
`pool_link`



---

---

## palloc\_fast

---

---

```
xmem void * palloc_fast( Pool_t * p );
```

### DESCRIPTION

Return next available free element from the given pool, which must be a root pool.

This is an assembler-only version of `palloc()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

`palloc_fast` does not perform any IPSET protection, parameter validation, or update the high-water mark. `palloc_fast` is a root function. The parameter must be passed in IX, and the returned element address is in HL.

### REGISTERS

Parameter in IX

Trashes F, BC, DE

Return value in HL, carry flag.

### EXAMPLE

```
ld ix,my_pool
lcall palloc_fast
jr c,.no_free
; HL points to element
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`. Pass this in IX.

### RETURN VALUE

C flag set: no free elements were available.

C flag clear (NC): HL points to an element.

If the pool is not linked, your application can use this element provided it does not write more than `p->elsize` bytes to it (this was the `elsize` parameter passed to `pool_init()`). If the pool is linked, you can write `p->elsize-4` bytes to it.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pfree_fast`, `pavail_fast`, `palloc`

---

---

## pavail

---

---

```
word pavail( Pool_t * p );
```

### DESCRIPTION

Return the number of elements that are currently available for allocation.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()` or `pool_xinit()`.

### RETURN VALUE

Number of elements available for allocation.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_xinit`, `phwm`, `pnel`

---

---

## pavail\_fast

---

---

```
xmem word pavail_fast( Pool_t * p );
```

### DESCRIPTION

Return the number of elements that are currently available for allocation.

This is an assembler-only version of `pavail()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

### REGISTERS

Parameter in IX

Trashes F, DE

Return value in HL, Z flag

### EXAMPLE

```
ld ix,my_pool
lcall pavail_fast
; HL contains number of available elements
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()` or `pool_xinit()`. This must be provided in the IX register.

### RETURN VALUE

Number of elements available for allocation. The return value is placed in HL. In addition, the 'Z' flag is set if there are no free elements.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_xinit`, `phwm`, `pnel`

---

---

## pccalloc

---

---

```
void * pccalloc( Pool_t * p );
```

### DESCRIPTION

Return next available free element from the given pool. Eventually, your application should return this element to the pool using `pfree()` to avoid memory leaks.

The element is set to all zero bytes before returning.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`.

### RETURN VALUE

Null: No free elements were available

Otherwise, pointer to an element. If the pool is not linked, your application must not write more than `p->elsize` bytes to the element (this was the `elsize` parameter passed to `pool_init()`). The application can write up to `(p->elsize-4)` bytes to the element if the pool is linked. (An element in root memory has 4 bytes of overhead when the pool is linked.)

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_init`, `palloc`, `pfree`, `phwm`, `pavail`

---

---

## **pfirst**

---

---

```
void * pfirst( Pool_t * p );
```

### **DESCRIPTION**

Get the first allocated element in a root pool. The pool **MUST** be set to being a linked pool using:

```
pool_link(p, <non-zero>)
```

Otherwise, the result is undefined.

### **PARAMETERS**

**p** Pool handle structure, as previously passed to `pool_init()`.

### **RETURN VALUE**

Null: There are no allocated elements

Otherwise, pointer to first (i.e., oldest) allocated element

### **LIBRARY**

POOL.LIB

### **SEE ALSO**

`pool_init`, `pool_link`, `palloc`, `pfree`, `plast`, `pnext`, `pprev`

---

---

## pfirst\_fast

---

---

```
xmem void * pfirst_fast( Pool_t * p );
```

### DESCRIPTION

Get the first allocated element in a root pool. The pool MUST be set to being a linked pool by using:

```
pool_link(p, <non-zero>);
```

Otherwise the results are undefined.

This is an assembler-only version of `pfirst()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

### REGISTERS

Parameter in IX

Trashes F, DE

Return value in HL, carry flag

### EXAMPLE

```
ld ix,my_pool
lcall pfirst_fast
jr c,.no_elems
; HL points to first element
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`. Pass this in the IX register.

### RETURN VALUE

C flag set, HL=0: There are no allocated elements.

C flag clear (NC): HL points to first element.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `pfirst`, `pnext_fast`

---

---

## pfree

---

---

```
void pfree( Pool_t * p, void * e );
```

### DESCRIPTION

Free an element that was obtained via `palloc()`. Note: if you free an element that was not allocated from this pool, or was already free, or was outside the pool, then your application will crash! You can detect most of these programming errors by defining the following symbols before `#use pool.lib`:

```
POOL_DEBUG
POOL_VERBOSE
```

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>palloc()</code> .
<b>e</b>	Element to free, which was returned from <code>palloc()</code> .

### RETURN VALUE

None

### LIBRARY

```
POOL.LIB
```

### SEE ALSO

```
pool_init, palloc, pcalloc, phwm, pavail
```

---

---

## pfree\_fast

---

---

```
xmem void pfree_fast( Pool_t * p, void * e );
```

### DESCRIPTION

Free an element that was previously obtained via `palloc()`.

This is an assembler-only version of `pfree()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

`pfree_fast` does not perform any IPSET protection or parameter validation. `pfree_fast` is a `xmem` function. The parameters must be passed in machine registers.

### REGISTERS

Parameters in IX, DE respectively

Trashes BC, DE, HL

### EXAMPLE

```
ld ix,my_pool
ld de,(element_addr)
lcall pfree_fast
```

### PARAMETERS

- |          |   |
|----------|---|
| <b>p</b> | Pool handle structure, as previously passed to <code>pool_alloc()</code> or <code>palloc_fast</code> . This must be in the IX register. |
| <b>e</b> | Element to free, which was returned from <code>palloc()</code> . This must be in the DE register.                                       |

### RETURN VALUE

None

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `palloc_fast`, `pavail_fast`, `pxfree_fast`



---

---

## phwm

---

---

```
word phwm( Pool_t * p );
```

### DESCRIPTION

Return the largest number of elements ever simultaneously allocated from the given pool, i.e., the pool high water mark.

You can use this function to help size a pool, since it may be difficult to determine the optimum number of elements without running a trial program.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()` or `pool_xinit()`.

### RETURN VALUE

Maximum number of elements ever allocated.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_xinit`, `pavail`

---

---

## pktXclose

---

---

```
void pktXclose( void ); /* X is A-F */
```

### DESCRIPTION

Disables serial port X. The functions `pktEclose()` and `pktFclose()` may be used with the Rabbit 3000 and Rabbit 4000.

### LIBRARY

PACKET.LIB

---

---

## pktXgetErrors

---

---

```
char pktXgetErrors( void ); /* X is A-F */
```

### DESCRIPTION

Gets a bit field with flags set for any errors that occurred on port X. These flags are then cleared, so that a particular error will only cause the flag to be set once.

The functions `pktEgetErrors()` and `pktFgetErrors()` may be used with the Rabbit 3000 and Rabbit 4000.

### RETURN VALUE

A bit field with flags for various errors. The errors along with their bit masks are as follows:

<code>PKT_BUFFEROVERFLOW</code>	<code>0x01</code>
<code>PKT_RXOVERRUN</code>	<code>0x02</code>
<code>PKT_PARITYERROR</code>	<code>0x04</code>
<code>PKT_NOBUFFER</code>	<code>0x08</code>

### LIBRARY

PACKET.LIB

---

---

## pktXinitBuffers

---

---

```
int pktXinitBuffers( int buf_count, int buf_size ); /* X is A-F */
```

### DESCRIPTION

Allocates extended memory for channel X receive buffers. This function should not be called more than once in a program. The total memory allocated is `buf_count*(buf_size + 2)` bytes.

The functions `pktEinitBuffers()` and `pktFinitBuffers()` may be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

<b>buf_count</b>	The number of buffers to allocate. Each buffer can store one received packet. Increasing this number allows for more pending packets and a larger latency time before packets must be processed by the user's program.
<b>buf_size</b>	The number of bytes each buffer can accommodate. This should be set to the size of the largest possible packet that can be expected.

### RETURN VALUE

1: Success, extended memory was allocated.  
0: Failure, no memory allocated, the packet channel cannot be used.

### LIBRARY

PACKET.LIB

---

---

## pktXopen

---

---

```
int pktXopen( long baud, int mode, char options, int (*test_packet)()
); /* X is A-F */
```

### DESCRIPTION

Opens serial port X. The functions `pktEopen()` and `pktFopen()` may be used with the Rabbit 3000 and Rabbit 4000.

The packet driver is meant to be used with a variety of transceiver hardware, so some functions must be defined by the user. Each of these functions, listed below, take no arguments and return nothing.

- `pktXinit()` - Initializes the communication hardware. Called inside `pktXopen()`. This function may be written in C. It will only be called once each time the packet driver is opened, so speed is not a major concern. This is where I/O pins should be configured and any other setup should be performed.
- `pktXrx()` - Sets the hardware to receive data. This function must be written in assembly. Any registers besides the 8-bit accumulator A must be preserved first, and restored before returning. This function is called when the driver switches from transmit to receive mode once there are no packets to send. This function is necessary for half-duplex connections and other types of shared bus schemes so that the transmitter can be disabled, allowing other nodes to use the lines.
- `pktXtx()` - Sets the hardware to transmit data. This function must be written in assembly. The same rules for register usage as for `pktXrx()` apply. This function is called whenever the driver switches from receive to transmit mode in response to an additional packet or packets being available for sending. A typical use of this function is to enable any necessary transmitter hardware.

See the sample program `Samples/PKTDEMO.C` for an example of how to write these user-supplied functions. See technical note TN213 “Rabbit Serial Port Software” for more information on the packet driver.

---

---

## pktXopen (cont'd)

---

---

### PARAMETERS

- baud** Bits per second of data transfer: minimum is 2400.
- mode** Type of packet scheme used, the options are:
- PKT\_GAPMODE
  - PKT\_9BITMODE
  - PKT\_CHARMODE
- options** Further specification for the packet scheme. The value of this depends on the mode used:
- gap mode - minimum gap size (in byte times)
  - 9-bit mode - type of 9-bit protocol
    - PKT\_RABBITSTARTBYTE
    - PKT\_LOWSTARTBYTE
    - PKT\_HIGHSTARTBYTE
  - char mode - character marking start of packet
- test\_packet** Pointer to a function that tests for completeness of a packet. The function should return 1 if the packet is complete, or 0 if more data should be read in. For gap mode the test function is not used and should be set to null.

### RETURN VALUE

- 1: The baud set on the rabbit is the same as the input baud.
- 0: The baud set on the rabbit does not match the input baud.

### LIBRARY

PACKET.LIB

---

---

## pktXreceive

---

---

```
int pktXreceive( void * buffer, int buffer_size ); /* X is A-F */
```

### DESCRIPTION

Gets a received packet, if there is one, from serial port X.

The functions `pktEreceive()` and `pktFreceive()` may be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

**buffer**            A buffer for the packet to be written into.

**buffer\_size**    Length of the data buffer.

### RETURN VALUE

>0: Number of bytes in the successfully received packet.

0: No new packet has been received.

-1: The packet is too large for the given buffer.

-2: A needed `test_packet` function is not defined.

### LIBRARY

PACKET.LIB

---

---

## pktXsend

---

---

```
int pktXsend( void *send_buffer, int buffer_length, char delay );  
/* X is A-F */
```

### DESCRIPTION

Initiates the sending of a packet of data using serial port X. This function will always return immediately. If there is already a packet being transmitted, this call will return 0 and the packet will not be transmitted, otherwise it will return 1.

`pktXsending()` checks if the packet is done transmitting. The system will be using the buffer until then.

The functions `pktEsend()` and `pktFsend()` may be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

<code>send_buffer</code>	The data to be sent
<code>buffer_length</code>	Length of the data buffer to transmit
<code>delay</code>	The number of byte times to delay before sending the data (0-255) This is used to implement protocol-specific delays between packets

### RETURN VALUE

1: The packet is going to be transmitted.  
0: There is already a packet transmitting, and the new packet was refused.

### LIBRARY

PACKET.LIB

---

---

## pktXsending

---

---

```
int pktXsending( void ); /* X is A-F */
```

### DESCRIPTION

Tests if a packet is currently being sent on serial port X. If `pktXsending()` returns true, the transmitter is busy and cannot accept another packet.

The functions `pktEsending()` and `pktFsending()` may be used with the Rabbit 3000 and Rabbit 4000.

### RETURN VALUE

- 1: A packet is being transmitted.
- 0: Port X is idle, ready for a new packet.

### LIBRARY

PACKET.LIB

---

---

## pktXsetParity

---

---

```
void pktXsetParity( char mode ); /* X is A-F */
```

### DESCRIPTION

Configures parity generation and checking. Can also configure for 2 stop bits.

The functions `pktEsetParity()` and `pktFsetParity()` may be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

- |             |   |
|-------------|---|
| <b>mode</b> | Code for mode of parity bit: <ul style="list-style-type: none"><li>• PKT_NOPARITY - no parity bit (8N1 format, default)</li><li>• PKT_OPARITY - odd parity (8O1 format)</li><li>• PKT_EPARITY - even parity (8E1 format)</li><li>• PKT_TWOSTOP - an extra stop bit (8N2 format)</li></ul> |
|-------------|---|

### LIBRARY

PACKET.LIB



---

---

## plast

---

---

```
void * plast( Pool_t * p );
```

### DESCRIPTION

Get the last allocated element in a root pool. The pool MUST be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`.

### RETURN VALUE

`null`: There are no allocated elements  
`!null`: Pointer to last, i.e., youngest, allocated element

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_init`, `pool_link`, `palloc`, `pfree`, `pfirst`

---

---

## plast\_fast

---

---

```
xmem void * plast_fast( Pool_t * p );
```

### DESCRIPTION

Get the last allocated element in a root pool. The pool MUST be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `plast()`.

\*\*\* Do \_not\_ call this function from C. \*\*\*

### Registers

Parameter in IX

Trashes F, DE

Return value in HL, carry flag

### Example

```
ld ix,my_pool
lcall plast_fast
jr c,.no_elems
; HL points to last element
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`. Pass this in IX register.

### RETURN VALUE

C flag set, HL=0: there are no allocated elements

C flag clear (NC): HL points to last element.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `plast`, `pprev_fast`

---

---

## pmovebetween

---

---

```
void * pmovebetween( Pool_t * p, void * e, void * d, void * f );
```

### DESCRIPTION

Atomically remove allocated element “e” and re-insert it between allocated elements “d” and “f.” “Atomically” means that the `POOL_IPSET` level is used to lock out other CPU contexts from altering the pool while this operation is in progress. In addition, “d” and “f” are checked to ensure that the following conditions still hold:

```
pprev(p, f) == d
```

and

```
pnext(p, d) == f
```

in other words, “f” follows “d.” This is useful since your application may have determined “d” and “f” some time ago, but in the meantime some other task may have re-ordered the queue or deleted these elements. In this case, the return value will be null. Your application should then re-evaluate the appropriate queue elements and retry this function.

The pool **MUST** be set to being a linked pool by using:

```
pool_link(p, <non-zero>)
```

Otherwise the results are undefined.

### PARAMETERS

- |          |   |
|----------|---|
| <b>p</b> | Pool handle structure, as previously passed to <code>pool_init()</code> .   |
| <b>e</b> | Address of element to move, obtained by, e.g., <code>plast()</code> . This must be an allocated element in the given pool; otherwise, the results are undefined. If null, then the last element is implied (i.e., whatever <code>plast()</code> would return). If there are no elements at all, or this parameter does not point to a valid allocated element, then the results are undefined (and probably catastrophic).<br><br>If <code>e == d</code> or <code>e == f</code> , then there is no action except to check whether “f” follows “d.” This parameter may refer to an unlinked (but allocated) element. |
| <b>d</b> | First reference element. The element “e” will be inserted after this element. On entry, it must be true that <code>pnext(p, d) == f</code> . Otherwise, null is returned. If this parameter is null, then “f” must point to the first element in the list, and “e” is inserted at the start of the list.  |

---

---

## pmovebetween (cont'd)

---

---

**f** Second reference element. The element “e” will be inserted before this element. On entry, it must be true that `pprev(p, f) == d`. Otherwise, null is returned. If this parameter is null, then “d” must point to the last element in the list, and “e” is inserted at the end of the list.

**Note:** If both “d” and “f” are null, then it must be true that there are no allocated elements in the linked list, and the element “e” is added as the only element in the list. This proviso only obtains when the element “e” is initially allocated from an empty pool with:

```
pool_link(p, POOL_LINKED_BY_APP)
```

The allocated element is not in the linked list of allocated elements.

### RETURN VALUE

Returns the parameter value “e,” unless “e” was null; in which case the value of `plast()`, if called at function entry, would be returned. If the initial conditions for “d” and “f” do not hold, then null is returned with no further action.

### EXAMPLES

```
void * d, * e, * f;

e = plast(p);                // element to move
f = pnext(p, d = pfirst(p)); // d, f are first 2 elements
pmovebetween(p, e, d, f);
```

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `plast`, `pfirst`, `pnext`, `pprev`, `preorder`

---

---

## pmovebetween\_fast

---

---

```
void * pmovebetween_fast( Pool_t *p, void *e, void *d, void *f );
```

### DESCRIPTION

See description under `pmovebetween()`. This is an assembler-callable version (do not call from C). It does not issue IPSET protection or check parameters.

REGISTERS: Parameters in IX, DE, BC, HL respectively

Trashes AF, BC, DE, BC', DE', HL'

Return value in HL, carry flag.

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>pool_init()</code> . Pass in IX register
<b>e</b>	Address of element to move. Pass in DE register.
<b>d</b>	The first reference element. Pass in BC register.
<b>f</b>	The second reference element. Pass in HL register.

### RETURN VALUE

In HL. Either set to “e” parameter, or 0. The carry flag is set if `HL==0`; otherwise it is clear.

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pmovebetween`

---

---

## pnel

---

---

```
word pnel( Pool_t * p );
```

### DESCRIPTION

Return the number of elements that are in the pool, both free and used. This includes elements appended using `pool_append()` etc.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()` or `pool_xinit()`.

### RETURN VALUE

Number of elements total

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_xinit`, `pavail`

---

---

## pnext

---

---

```
void * pnext( Pool_t * p, void * e );
```

### DESCRIPTION

Get the next allocated element in a root pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

You can easily iterate through all of the allocated elements of a root pool using the following construct:

```
void * e;
Pool_t * p;
for (e = pfirst(p); e; e = pnext(p, e)) {
    ...
}
```

### PARAMETERS

- |          |  |
|----------|--|
| <b>p</b> | Pool handle structure, as previously passed to <code>pool_init()</code> .  |
| <b>e</b> | Previous element address, obtained by, e.g., <code>pfirst()</code> . This must be an allocated element in the given pool; otherwise, the results are undefined. Be careful when iterating through a list and deleting elements using <code>pfree()</code> : once the element is deleted, it is no longer valid to pass its address to this function.<br><br>If this parameter is null, then the result is the same as <code>pfirst()</code> . This ensures the invariant <code>pnext(p, pprev(p, e)) == e</code> . |

### RETURN VALUE

- null: There are no more elements
- !null: Pointer to next allocated element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `palloc`, `pfree`, `pfirst`, `pprev`

---

---

## pnext\_fast

---

---

```
xmem void * pnext_fast( Pool_t * p, void * e );
```

### DESCRIPTION

Get the next allocated element in a root pool. The pool MUST be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `pnext()`.

\*\*\* Do \_not\_ call this function from C. \*\*\*

### Registers

Parameters in IX, DE respectively

Trashes F, DE

Return value in HL, carry flag

### Example

```
ld ix,my_pool
ld de,(current_element)
lcall pnext_fast
jr c,.no_more_elems
; HL points to the next allocated element
```

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>pool_init()</code> . Pass this in IX register.
<b>e</b>	Current element, address in DE register. See <code>pnext()</code> for a full description.

### RETURN VALUE

C flag set, HL=0: There are no more elements

C flag clear (NC): HL points to next element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `palloc`, `pfree`, `pfirst`, `pprev`



---

---

## poly

---

---

```
float poly( float x, int n, float c[] );
```

### DESCRIPTION

Computes polynomial value by Horner's method. For example, for the fourth-order polynomial  $10x^4 - 3x^2 + 4x + 6$ , `n` would be 4 and the coefficients would be

```
c[4] = 10.0  
c[3] = 0.0  
c[2] = -3.0  
c[1] = 4.0  
c[0] = 6.0
```

### PARAMETERS

<b>x</b>	Variable of the polynomial.
<b>n</b>	The order of the polynomial
<b>c</b>	Array containing the coefficients of each power of <code>x</code> .

### RETURN VALUE

The polynomial value.

### LIBRARY

`MATH.LIB`

---

---

## pool\_append

---

---

```
int pool_append( Pool_t * p, void * base, word nel );
```

### DESCRIPTION

Add another root memory area to an existing pool. It is assumed that the element size is the same as the element size of the existing pool.

The data area does not have to be contiguous with the existing data area, but it must be `nel*elsize` bytes long (where `elsize` is the element size of the existing pool, and `nel` is the parameter to this function).

The total pool size must obey the constraints documented with `pool_init()`.

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>pool_init()</code> .
<b>base</b>	Base address of the root data memory area to append to this pool. This must be <code>nel*elsize</code> bytes long. Typically, this would be a static (global) array.
<b>nel</b>	Number of elements in the memory area. The sum of <code>nel</code> and the current number of elements must not exceed 32767.

### RETURN VALUE

Currently always zero. If you define the macro `POOL_DEBUG`, then parameters are checked. If the parameters look bad, then an exception is raised. You can define `POOL_VERBOSE` to get `printf()` messages.

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_init`

---

---

## pool\_init

---

---

```
int pool_init( Pool_t * p, void * base, word nel, word elsize );
```

### DESCRIPTION

Initialize a root memory pool. A pool is a linked list of fixed-size blocks taken from a contiguous area. You can use pools instead of `malloc()` when fixed-size blocks are all that is needed. You can have several pools, with different size blocks. Using memory pools is very efficient compared with more general functions like `malloc()`. (There is currently no `malloc()` implementation with Dynamic C.)

This function should only be called once, at program startup time, for each pool to be used.

Note: the product of `nel` and `elsize` must be less than 65535 (however, this will usually be limited further by the actual amount of root memory available).

After calling this function, your application must not change any of the fields in the `Pool_t` structure.

### PARAMETERS

<b>p</b>	Pool handle structure. This is allocated by the caller, but this function will initialize it. Normally, this would be allocated in static memory by declaring a global variable of type <code>Pool_t</code> .
<b>base</b>	Base address of the root data memory area to be managed in this pool. This must be <code>nel*elsize</code> bytes long. Typically, this would be a static (global) array.
<b>nel</b>	Number of elements in the memory area. 1..32767
<b>elsize</b>	Size of each element in the memory area. 2..32767

### RETURN VALUE

Currently always zero. If you define the macro `POOL_DEBUG`, then parameters are checked. If the parameters look bad, then an exception is raised. You can define `POOL_VERBOSE` to get `printf()` messages.

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_xinit`, `palloc`, `pcalloc`, `pfree`, `phwm`, `pavail`

---

---

## pool\_link

---

---

```
int pool_link( Pool_t * p, int link );
```

### DESCRIPTION

Tell the specified pool to maintain a doubly-linked list of allocated elements.

This function should only be called when the pool is completely free; i.e.,

```
pavail() == pnel()
```

### PARAMETERS

- |             |   |
|-------------|---|
| <b>p</b>    | Pool handle structure, as previously passed to <code>pool_init()</code> or <code>pool_xinit()</code> .  |
| <b>link</b> | Must be one of the following: <ul style="list-style-type: none"><li>• <code>POOL_NOT_LINKED (0)</code>: the pool is not to be linked.</li><li>• <code>POOL_LINKED_AUTO (1)</code>: the pool is linked, and newly allocated elements are always added at the end of the list.</li><li>• <code>POOL_LINKED_BY_APP (2)</code>: the pool is linked, but newly allocated elements are not added to the list. The application must call <code>preorder()</code> or <code>pmovebetween()</code> to insert the element. This option is only available for root pools.</li></ul> |

**WARNING:** if you set the `POOL_LINKED_BY_APP` option, then the allocated element must NOT be passed to any other pool API function except for `pfree()`, `preorder()` (as the “e” parameter) or `pmovebetween()` (as the “e” parameter). After calling `preorder()` or `pmovebetween()`, then it is safe to pass this element to all appropriate functions.

### RETURN VALUE

Currently always zero. If you define the macro `POOL_DEBUG`, then parameters are checked. If the parameters look bad, then an exception is raised. You can define `POOL_VERBOSE` to get `printf()` messages.

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_init`, `pool_xinit`, `pavail`

---

---

## pool\_xappend

---

---

```
int pool_xappend( Pool_t * p, long base, word nel );
```

### DESCRIPTION

Add another xmem memory area to an existing pool. It is assumed that the element size is the same as the element size of the existing pool.

The data area does not have to be contiguous with the existing data area, but it must be `nel*elsize` bytes long (where `elsize` is the element size of the existing pool, and `nel` is the parameter to this function).

The total pool size must obey the constraints documented with `pool_xinit()`.

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>pool_xinit()</code> .
<b>base</b>	Base address of the xmem data memory area to append to this pool. This must be <code>nel*elsize</code> bytes long. Typically, this would be an area allocated using <code>xalloc()</code> .
<b>nel</b>	Number of elements in the memory area. 1..65534. The sum of this and the current number of elements must not exceed 65535.

### RETURN VALUE

Currently always zero. If you define the macro `POOL_DEBUG`, then parameters are checked. If the parameters look bad, then an exception is raised. You can define `POOL_VERBOSE` to get `printf()` messages.

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_xinit`

---

---

## pool\_xinit

---

---

```
int pool_xinit( Pool_t * p, long base, word nel, word elsize );
```

### DESCRIPTION

Initialize an xmem memory pool. A pool is a linked list of fixed-size blocks taken from a contiguous area. You can use pools instead of malloc() when fixed-size blocks are all that is needed. You can have several pools, with different size blocks. Using memory pools is very efficient compared with more general functions like malloc(). (There is currently no malloc() implementation with Dynamic C.)

This function should only be called once, at program startup time, for each pool to be used.

After calling this function, your application must not change any of the fields in the Pool\_t structure.

### PARAMETERS

<b>p</b>	Pool handle structure. This is allocated by the caller, but this function will initialize it. Normally, this would be allocated in static memory by declaring a global variable of type Pool_t.
<b>base</b>	Base address of the xmem data memory area to be managed in this pool. This must be nel*elsize bytes long. Typically, this would be an area allocated by xalloc() when your program starts.
<b>nel</b>	Number of elements in the memory area. 1..65535
<b>elsize</b>	Size of each element in the memory area. 4..65535

### RETURN VALUE

Currently always zero. If you define the macro POOL\_DEBUG, then parameters are checked. If the parameters look bad, then an exception is raised. You can define POOL\_VERBOSE to get printf() messages.

### LIBRARY

POOL.LIB

### SEE ALSO

pool\_init, pxalloc, pxcalloc, pxfree, phwm, pavail

---

---

## pow

---

---

```
float pow( float x, float y );
```

### DESCRIPTION

Raises x to the yth power.

### PARAMETERS

<b>x</b>	Value to be raised
<b>y</b>	Exponent

### RETURN VALUE

x to the yth power

### LIBRARY

MATH.LIB

### SEE ALSO

exp, pow10, sqrt

---

---

## pow10

---

---

```
float pow10( float x );
```

### DESCRIPTION

10 to the power of x.

### PARAMETERS

<b>x</b>	Exponent
----------	----------

### RETURN VALUE

10 raised to power x

### LIBRARY

MATH.LIB

### SEE ALSO

pow, exp, sqrt

---

---

## powerspectrum

---

---

```
void powerspectrum( int * x, int N, * int blockexp );
```

### DESCRIPTION

Computes the power spectrum from a complex spectrum according to

$$\text{Power}[k] = (\text{Re } X[k])^2 + (\text{Im } X[k])^2$$

The N-point power spectrum replaces the N-point complex spectrum. The power of each complex spectral component is computed as a 32-bit fraction. Its more significant 16-bits replace the imaginary part of the component; its less significant 16-bits replace the real part.

If the complex input spectrum is a positive-frequency spectrum computed by `fftrealm()`, the imaginary part of the  $X[0]$  term (stored `x[1]`) will contain the real part of the *fmax* term and will affect the calculation of the dc power. If the dc power or the *fmax* power is important, the *fmax* term should be retrieved from `x[1]` and `x[1]` set to zero before calling `powerspectrum()`.

The power of the *k*th term can be retrieved via

$$P[k] = *(\text{long}*) \&x[2k] * 2^{\text{blockexp}}.$$

The value of `blockexp` is first doubled to reflect the squaring operation applied to all elements in array `x`. Then it is further increased by 1 to reflect an inherent division by two that occurs during the squaring operation.

### PARAMETERS

<b>x</b>	Pointer to N-element array of complex fractions.
<b>N</b>	Number of complex elements in array <code>x</code> .
<b>blockexp</b>	Pointer to integer block exponent.

### LIBRARY

FFT.LIB

### SEE ALSO

`fftcplx`, `fftcplxinv`, `fftrealm`, `fftrealminv`, `hanncplx`, `hannreal`



---

---

## pprev

---

---

```
void * pprev( Pool_t * p, void * e );
```

### DESCRIPTION

Get the previously allocated element in a root pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

You can easily iterate through all of the allocated elements of a root pool using the following construct:

```
void * e;
Pool_t * p;

for (e = plast(p); e; e = pprev(p, e)) {
    ...
}
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`.

**e** Previous element address, obtained by, e.g., `plast()`. This must be an allocated element in the given pool; otherwise, the results are undefined. Be careful when iterating through a list and deleting elements using `pfree()`: once the element is deleted, it is no longer valid to pass its address to this function. If this parameter is null, then the result is the same as `plast()`. This ensures the invariant

$$\text{pprev}(p, \text{pnext}(p, e)) == e$$

### RETURN VALUE

null: There are no more elements

!null: Pointer to previous allocated element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `palloc`, `pfree`, `plast`, `pnext`

---

---

## pprev\_fast

---

---

```
xmem void * pprev_fast( Pool_t * p, void * e );
```

### DESCRIPTION

Get the previous allocated element in a root pool. The pool MUST be set to being a linked pool by using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `pprev()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

### Registers

Parameters in IX, DE respectively

Trashes F, DE

Return value in HL, carry flag

### Example

```
ld ix,my_pool
ld de,(current_element)
lcall pprev_fast
jr c,.no_more_elems
; HL points to previously allocated element
```

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>pool_init()</code> . Pass this in IX register.
<b>e</b>	Current element, address in DE register. See <code>pprev()</code> for fuller description.

### RETURN VALUE

C flag set, HL=0: There are no more elements  
C flag clear (NC): HL points to previous element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `palloc`, `pprev`

---

---

## pputlast

---

---

```
void * pputlast(Pool_t * p, void * e);
```

### DESCRIPTION

Atomically remove allocated element “e” and re-insert it at the end of the allocated list. “Atomically” means that the `POOL_IPSET` level is used to lock out other CPU contexts from altering the pool while this operation is in progress.

This is equivalent to:

```
pmovebetween(p, e, plast(p), NULL);
```

but is considerably faster.

A common use for this function is to insert an element allocated when the `POOL_LINKED_BY_APP` attribute is set for the pool, at the end of the allocated list. This is useful when, say, an ISR allocates and uses a buffer without placing it on the allocated list. Only when the buffer is complete does the ISR use this function to place it on the queue for reading by the main application.

The pool **MUST** be set to being a linked pool by using:

```
pool_link(p, <non-zero>);
```

otherwise the results are undefined.

### PARAMETERS

<b>p</b>	Pointer to pool handle structure, as previously passed to <code>pool_init()</code> .
<b>e</b>	Address of element to move. If <code>NULL</code> , then this function behaves as <code>plast()</code> .

### RETURN VALUE

Same as the “e” parameter, unless “e” is `NULL` in which case the existing last element is returned as per `plast()`.

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pmovebetween`, `pool_link`

---

---

## pputlast\_fast

---

---

```
void * pputlast_fast(Pool_t * p, void * e);
```

### DESCRIPTION

See description under `pputlast()`. This is an assembler-callable version (do not call from C). It does not issue IPSET protection or check parameters.

#### Registers:

Parameters in IX (“p”) and DE (“e”)  
Trashes F, DE, BC  
Return value in HL

### PARAMETERS

<b>p</b>	Pointer to pool handle structure, as previously passed to <code>pool_init()</code> . Pass in IX register
<b>e</b>	Address of element to move. Pass in DE register. If NULL, then this function behaves as <code>plast_fast()</code> .

### RETURN VALUE

In HL. Same as the “e” parameter, unless “e” is NULL in which case the existing last element is returned as per `plast_fast()`.

### LIBRARY

POOL.LIB

### SEE ALSO

`pmovebetween`, `pool_link`

---

---

## premain

---

---

```
void premain( void );
```

### DESCRIPTION

Dynamic C calls `premain` to start initialization functions such as `VdInit`. The final thing `premain` does is call `main`. This function should never be called by an application program. It is included here for informational purposes only.

### LIBRARY

PROGRAM.LIB

---

---

## preorder

---

---

```
void * preorder( Pool_t *p, void *e, void *where, word options );
```

### DESCRIPTION

Atomically remove allocated element “e” and re-insert it before or after element “where.” “Atomically” means that the `POOL_IPSET` level is used to lock out other CPU contexts from altering the pool while this operation is in progress.

The pool **MUST** be set to being a linked pool by using:

```
pool_link(p, <non-zero>)
```

Otherwise the results are undefined.

### PARAMETERS

- |                |   |
|----------------|---|
| <b>p</b>       | Pool handle structure, as previously passed to <code>pool_init()</code> .   |
| <b>e</b>       | Address of element to move, obtained by e.g., <code>plast()</code> . This must be an allocated element in the given pool; otherwise, the results are undefined. If null, then the last element is implied (i.e., whatever <code>plast()</code> would return). If there are no elements at all, or this parameter does not point to a valid allocated element, then the results are undefined (and probably catastrophic).   |
| <b>where</b>   | The reference element. The element “e” will be inserted before or after this element, depending on the options parameter. If <code>e==where</code> , then there is no action. If this parameter is null, then the reference element is assumed to be the first element (i.e., whatever <code>pfirst()</code> would return). If there are no elements at all, or this parameter does not point to a valid allocated element, then the results are undefined (and probably catastrophic). |
| <b>options</b> | Option flags. Currently, the only options are:<br><code>POOL_INSERT_BEFORE</code><br><code>POOL_INSERT_AFTER</code><br>which specifies whether “e” is to be inserted before or after “where.”   |

---

---

## preorder (cont'd)

---

---

### RETURN VALUE

Returns the parameter value “e” unless “e” was null, in which case the value of `plast()`, when called at function entry, would be returned.

**IMPORTANT:** If null is returned, that means that some other task (context, or ISR) modified the linked list while this operation was in progress. In this case, the application should call this function again with the same parameters, since this operation will NOT have completed. This would be a rare occurrence; however, multitasking applications should handle this case correctly.

### EXAMPLES

```
void * r;
void * s;

s = pnext(p, pfirst(p);          // s is second element
r = plast(p);                   // r is last element
preorder(p, s, r, POOL_INSERT_AFTER);

// If s != r, then s will become the new last element. You can use null
// parameters to perform the common case of moving the last element
// to the head of the list:
preorder(p, NULL, NULL, POOL_INSERT_BEFORE);

// which is identical to:
preorder(p, plast(p), pfirst(p), POOL_INSERT_BEFORE);
```

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pool_link`, `plast`, `pfirst`, `pnext`, `pprev`, `pmovebetween`

---

---

## printf

---

---

```
int printf( char *fmt, ... );
```

### DESCRIPTION

This function is similar to `sprintf()`, but outputs the formatted string to the Stdio window. Prior to Dynamic C 7.25, `printf()` would work only with the controller in program mode connected to a PC running Dynamic C. As of Dynamic C 7.25, it is possible to redirect `printf()` output to a serial port during run mode by defining a macro to specify the serial port. See the sample program `SAMPLES/STDIO_SERIAL.C` for more information.

See below for the complete list of Dynamic C Conversion Specifiers.

The user should make sure that:

- there are enough arguments after `fmt` to fill in the format parameters in the format string
- the types of arguments after `fmt` match the conversion specifiers in `fmt`

The macro `STDIO_DISABLE_FLOATS` can be defined if it is not necessary to format floating point numbers. If this macro is defined, `%e`, `%f` and `%g` will not be recognized. This can save thousands of bytes of code space.

The macro `STDIO_ENABLE_LONG_STRINGS` can be defined if it is necessary to print strings to the Stdio window that are longer than the default of 127 bytes. Without defining this macro, such strings are truncated. The drawback of defining this macro is that if it is defined in a multi-tasking application where more than one task is utilizing `printf` and at least one of the tasks is printing strings longer than 127 bytes, the user must ensure that calls to `printf` are serialized via a semaphore or similar means. If calls to `printf` are not serialized under these conditions, `printf` output from the different tasks may be interleaved in the Stdio window.

**Note:** this function is task reentrant and it has a 128 byte buffer.

### PARAMETERS

<code>fmt</code>	String to be formatted.
<code>...</code>	Format arguments.

### RETURN VALUE

Number of characters written

### LIBRARY

`STDIO.LIB`

### SEE ALSO

`sprintf`

---

---

## printf (cont'd)

---

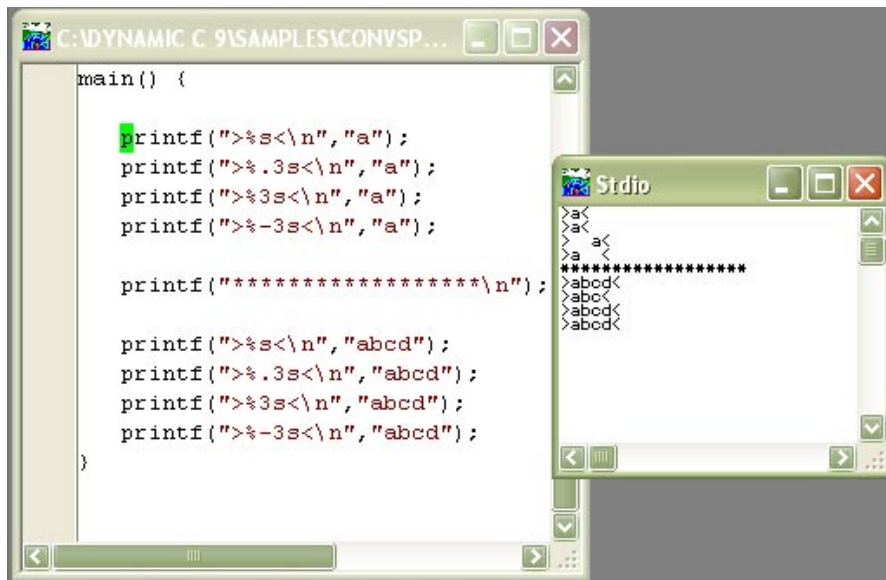
---

### DYNAMIC C CONVERSION SPECIFIERS

%s - string  
%ls - null terminated string in xmem  
%d - signed decimal  
%u - unsigned decimal  
%f - float  
%e - exponential  
%g - floating point, same as %f or %e depending upon value and precision  
%p - pointer  
%lp - pointer  
%x - hexadecimal, result in lowercase  
%X - hexadecimal, same as %x but result in uppercase  
%c - single character

%s - string

The precision specifier (the number between “%” and “s”) determines the maximum number of characters to display.



The screenshot shows a C program in a text editor window titled "C:\DYNAMIC C 9\SAMPLES\CONVSP...". The code defines a main function with several printf statements. The first four lines use the string "a" with different format specifiers: "%s", "%.3s", "%3s", and "%-3s". The next line prints a line of asterisks. The final four lines use the string "abcd" with the same four format specifiers. A separate "Stdio" window shows the output of the program, which matches the code: "a", "a", "a", "a", a line of asterisks, "abcd", "abcd", "abcd", "abcd".

```
main() {  
    printf(">%s<\n", "a");  
    printf(">%.3s<\n", "a");  
    printf(">%3s<\n", "a");  
    printf(">%-3s<\n", "a");  
  
    printf("*****\n");  
  
    printf(">%s<\n", "abcd");  
    printf(">%.3s<\n", "abcd");  
    printf(">%3s<\n", "abcd");  
    printf(">%-3s<\n", "abcd");  
}
```

As shown in the screenshot above, a value to the right of “.” causes the string to be displayed with that number of characters, ignoring extra characters. A value by itself or to the left of “.” causes padding. Negative values cause the string to be left justified, with spaces added to the right if necessary. Positive values cause the string to be right justified, with spaces added to the left if necessary.



---

---

## printf (cont'd)

---

---

**%ls** - null terminated string in xmem

This conversion specifier is identical to “%s” but the strings come from extended memory instead of root memory.

```
xdata mystring {"Now is the time."};
printf("%ls", mystring);          // Now is the time.
```

**%d** - signed decimal

Width specifier **l**: short values must not include **l**; without **l**, long values are treated as short

Precision specifier **n**: includes **'l'** and if necessary treats argument as signed

```
short n;
n = 30000;

printf("%d", n);          // 30000
printf("%5d", n);        // 30000
printf("%6d", n);        // 30000
printf("%4d", n);        // ****

unsigned short n;
n = 40000;

printf("%d", n);          // -25536
printf("%6d", n);        // -25536
printf("%7d", n);        // -25536
printf("%5d", n);        // *****

long n;
n = 300000;

printf("%ld", n);        // 300000
printf("%7ld", n);       // 300000
```

**%u** - unsigned decimal

Width specifier **l**: long values must include **l**, short values must not:

Precision specifier **n**: includes **'l'** if necessary treats argument as if it were unsigned

```
short n;
n = -25536;
printf("%u", n);          // 40000

unsigned short n;
n = 40000;
printf("%d", n);          // 40000
```

---

---

## printf (cont'd)

---

---

### %f - float

Width specifier *l* is ignored for Dynamic C float and double (both 4 bytes)

Precision specifier *n.d*: *n* is the total width including '*'* and '*'*'; if *n* is zero or is omitted, it is ignored and only *d* is used.

```
float f;
f = -88.8888;

printf("%f", f);           // -88.888801
printf("%10f", f);        // -88.888801
printf("%9f", f);         // *****
printf("%.0f", f);        // -89
printf("%.3f", f);        // -88.889
printf("%.0f", f);        // -88.889
printf("%.0.3f", f);      // -88.889
printf("%.7.3f", f);      // -88.889
printf("%.8.3f", f);      // -88.889
printf("%.6.3f", f);      // *****
```

### %e - exponential

Width specifier *l* is ignored for Dynamic C float and double (both 4 bytes)

Precision specifier *n.d*: *n* is the total width excluding exponent sign; if *n* is zero or is omitted, it is ignored and only *d* is used; if *n* larger than width, the result is not padded. *d* is decimal places of *n.nnn..E[+/-]nn* format

```
float f;
f = -88.8888;

printf("%e\n", f);        // -8.888880E+01
printf("%13e\n", f);     // -8.888880E+01
printf("%12e\n", f);     // -8.888880E+01
printf("%.0e\n", f);     // -9E+01
printf("%.1e\n", f);     // -8.9E+01
printf("%.3e\n", f);     // -8.889E+01
printf("%.0.3e\n", f);   // -8.889E+01
printf("%.9.3e\n", f);   // -8.889E+01
printf("%.15.3e\n", f);  // -8.889E+01
printf("%.8.3e\n", f);   // *****
printf("%.8.3e\n", -f);  // 8.889E+01
```

---

---

## printf (cont'd)

---

---

**%g** - floating point

(Same as %f or %e depending upon value and precision.)

```
float f, g, h;
f = -888.8888;
g = 888888.0
g = 8888880.0

printf("%g\n", g);           // 888888.0
printf("%g\n", h);           // 8.888880E+06
printf("%g\n", f);           // -888.888790
printf("%13g\n", f);         // -888.888790
printf("%12g\n", f);         // -888.888790
printf("%.0g\n", f);         // -8.9E+02
printf("%.1g\n", f);         // -8.9E+02
printf("%.2g\n", f);         // -8.89E+02
printf("%.3g\n", f);         // -888.889
printf("%7.3g\n", f);        // *****
printf("%0.3g\n", f);        // -888.889
printf("%9.3g\n", f);        // -888.889
printf("%15.3g\n", f);       // -888.889
printf("%8.3g\n", f);        // -888.889
printf("%8.3g\n", -f);       // 888.889
```

**%p** - pointer

Specifies a 16-bit logical pointer.

```
int i, *iptr;

i = 0;
ptr = &i;

printf("%p\n", ptr);        // prints value of ptr in hex.
                             // logical memory location of i
```

**%lp** - pointer

Specifies a 32-bit physical pointer.

```
long i, *iptr;

i = 0;
ptr = &i;
printf("%lp\n", ptr);       // prints value of ptr in hex.
                             // physical memory location of i
```

---

---

## printf (cont'd)

---

---

`%x` - hexadecimal

Result in lowercase

Width specifier `l`: short values must not include `l`; without `l`, long values are treated as short

Precision specifier `n`: must be at least as large as total width; treats argument as if it were unsigned

```
short n;
n = 30000;

printf("%x", n);           //7530
printf("%5x", n);         // 7530
printf("%6x", n);         // 7530
printf("%3x", n);         // ***

unsigned short n;
n = 40000;
printf("%x", n);           // 9c40

long m, n;
m = -25536;
n = 0x10000 + 0xabc;

printf("%x\n", m);        // 9c40
printf("%x\n", z);        // abc
```

`%X` - hexadecimal

Same as `%x` except the result is in uppercase.

`%c` - single character

Precision specifier `n` is ignored for `%c`; treats argument as if it were `char`

```
long n;
n = 0x10000 + 0x100 + 'A';
printf("%0c", n);         // A

short n;
n = 0x100 + 'A';
printf("%0c", n);         // A

char n;
n = 'A';
printf("%0c", n);         // A
```

Not supported:

`%o` - octal

`%E` - same as `%e`, result uppercase (the result is always in uppercase in Dynamic C)

`%G` - same as `%g`, result uppercase (the result is always in uppercase in Dynamic C)

---

---

## putchar

---

---

```
void putchar( int ch );
```

### DESCRIPTION

Puts a single character to Stdout. The user should make sure only one process calls this function at a time.

### PARAMETERS

**ch**                      Character to be displayed.

### LIBRARY

STDIO.LIB

### SEE ALSO

puts, getchar

---

---

## puts

---

---

```
int puts( char * s );
```

### DESCRIPTION

This function displays the string on the stdio window in Dynamic C. The Stdio window is responsible for interpreting any escape code sequences contained in the string. Only one process at a time should call this function.

### PARAMETERS

**s**                        Pointer to string argument to be displayed.

### RETURN VALUE

1: Success.

### LIBRARY

STDIO.LIB

### SEE ALSO

putchar, gets

---

---

## pwm\_init

---

---

```
unsigned long pwm_init( unsigned long frequency );
```

### DESCRIPTION

Sets the base frequency for the pulse width modulation (PWM) and enables the PWM driver on all four channels. The base frequency is the frequency without pulse spreading. Pulse spreading (see `pwm_set()`) will increase the frequency by a factor of 4.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### PARAMETER

**frequency**      Requested frequency (in Hz)

### RETURN VALUE

The actual frequency that was set. This will be the closest possible match to the requested frequency.

### LIBRARY

PWM.LIB (was in R3000.LIB prior to DC 10)

---

---

## pwm\_set

---

---

```
int pwm_set( int channel, int duty_cycle, int options );
```

### DESCRIPTION

Sets a duty cycle for one of the pulse width modulation (PWM) channels. The duty cycle can be a value from 0 to 1024, where 0 is logic low the whole time, and 1024 is logic high the whole time. Option flags are used to enable features on an individual PWM channel. Bit masks for these are:

- `PWM_SPREAD` - sets pulse spreading. The duty cycle is spread over four separate pulses to increase the pulse frequency.
- `PWM_OPENDRAIN` - sets the PWM output pin to be open-drain instead of a normal push-pull logic output.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

<code>channel</code>	channel(0 to 3)
<code>duty_cycle</code>	value from 0 to 1024
<code>options</code>	combination of optional flags (see above)

### RETURN VALUE

- 0: Success.
- 1: Error, an invalid channel number is used.
- 2: Error, requested `duty_cycle` is invalid.

### LIBRARY

`PWM.LIB` (was in `R3000.LIB` prior to DC 10)

---

---

## pxalloc

---

---

```
long pxalloc( Pool_t * p );
```

### DESCRIPTION

Return next available free element from the given pool. Eventually, your application should return this element to the pool using `pfree()` to avoid memory leaks.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_xinit()`.

### RETURN VALUE

0: No free elements are available.

!0: Physical (xmem address) of an element. If the pool is not linked, your application can use this element provided it does not write more than `p->elsize` bytes to it (this was the `elsize` parameter passed to `pool_xinit()`). If the pool is linked, you can write up to `(p->elsize-8)` bytes to it. (Each element has 8 bytes of overhead when the pool is linked.)

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pxcalloc`, `pxfree`, `phwm`, `pavail`



---

---

## pxalloc\_fast

---

---

```
xmem long pxalloc_fast( Pool_t * p );
```

### DESCRIPTION

Return next available free element from the given pool. Eventually, your application should return this element to the pool using `pxfree()` to avoid memory leaks.

This is an assembler-only version of `pxalloc()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

`pxalloc_fast` does not perform any IPSET protection, parameter validation, or update the high-water mark. `pxalloc_fast` is a root function. The parameter must be passed in IX, and the returned element address is in BCDE.

### REGISTERS

Parameter in IX

Trashes AF, HL

Return value in BCDE, carry flag.

### EXAMPLE

```
ld ix,my_pool
lcall pxalloc_fast
jr c,.no_free
; BCDE points to element
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()` Pass this in the IX register.

### RETURN VALUE

C flag set: No free elements are available. (BCDE is undefined in this case.)

NC flag: BCDE points to an element If the pool is not linked, your application must not write more than `p->elsize` bytes to it (this was the `elsize` parameter passed to `pool_xinit()`). If the pool is linked, you can write `(p->elsize-8)` bytes to it. (An element has 8 bytes of overhead when the pool is linked.)

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pxfree_fast`, `pavail_fast`, `pxalloc`

---

---

## pxcalloc

---

---

```
long pxcalloc( Pool_t * p );
```

### DESCRIPTION

Return next available free element from the given pool. Eventually, your application should return this element to the pool using `pxfree()` to avoid memory leaks.

The element is set to all zero bytes before returning.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_xinit()`.

### RETURN VALUE

0: No free elements are available.

!0: Physical (xmem address) of an element. If the pool is not linked, your application must not write more than `p->elsize` bytes to it (this was the `elsize` parameter passed to `pool_xinit()`). The application can write up to `(p->elsize-8)` bytes to the element if the pool is linked. (An element has 8 bytes of overhead when the pool is linked.)

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pxalloc`, `pxfree`, `phwm`, `pavail`

---

---

## pxfirst

---

---

```
long pxfirst( Pool_t * p );
```

### DESCRIPTION

Get the first allocated element in an xmem pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_xinit()`.

### RETURN VALUE

0: There are no allocated elements  
!0: Pointer to first, i.e., oldest, allocated element.

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxalloc`, `pxfree`, `pxlast`, `pxnext`, `pxprev`

---

---

## pxfirst\_fast

---

---

```
xmem long pxfirst_fast( Pool_t * p );
```

### DESCRIPTION

Get the first allocated element in an xmem pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `pxfirst()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

### Registers

Parameter in IX

Trashes F, HL

Return value in BCDE, carry flag

### Example

```
ld ix,my_pool
lcall pxfirst_fast
jr c,.no_elems
; BCDE points to first element
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_init()`. Pass this in IX register.

### RETURN VALUE

C flag set: There are no allocated elements

C flag clear (NC): BCDE points to first element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxfirst`, `pxnext_fast`

---

---

## pxfree

---

---

```
void pxfree( Pool_t * p, long e );
```

### DESCRIPTION

Free an element that was previously obtained via `pxalloc()`.

Note: if you free an element that was not allocated from this pool, or was already free, or was outside the pool, then your application will crash! You can detect most of these programming errors by defining the following symbols before `#use pool.lib`:

```
POOL_DEBUG
POOL_VERBOSE
```

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>pxalloc()</code> .
<b>e</b>	Element to free, which was returned from <code>pxalloc()</code> .

### RETURN VALUE

`null`: There are no more elements  
`!null`: Pointer to previous allocated element

### LIBRARY

```
POOL.LIB
```

### SEE ALSO

```
pool_xinit, pxalloc, pxcalloc, phwm, pavail
```

---

---

## pxfree\_fast

---

---

```
xmem void pxfree_fast( Pool_t * p, long e );
```

### DESCRIPTION

Free an element that was previously obtained via `pxalloc()`. This is an assembler-only version of `pxfree()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

`pxfree_fast` does not perform any IPSET protection or parameter validation.  
`pxfree_fast` is an xmem function. The parameters must be passed in machine registers.

### Registers

Parameters in IX, BCDE respectively  
Trashes AF, BC, DE, HL

### Example

```
ld ix,my_pool
ld de,(element_addr)
ld bc,(element_addr+2)
lcall pxfree_fast
```

### PARAMETERS

<b>p</b>	Pool handle structure, as previously passed to <code>palloc()</code> or <code>palloc_fast</code> . This must be in the IX register.
<b>e</b>	Element to free, which was returned from <code>palloc()</code> . This must be in the BCDE register (physical address)

### RETURN VALUE

`null`: There are no more elements  
`!null`: Pointer to previous allocated element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_init`, `pxalloc_fast`, `pavail_fast`, `pfree_fast`

---

---

## pxlast

---

---

```
long pxlast( Pool_t * p );
```

### DESCRIPTION

Get the last allocated element in an xmem pool. The pool MUST be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_xinit()`.

### RETURN VALUE

0: There are no allocated elements  
!0: Pointer to last, i.e., youngest, allocated element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxalloc`, `pxfree`, `pxfirst`

---

---

## pxlast\_fast

---

---

```
xmem long pxlast_fast( Pool_t * p );
```

### DESCRIPTION

Get the last allocated element in an xmem pool. The pool MUST be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `pxlast()`.

\*\*\* Do \_not\_ call this function from C. \*\*\*

### Registers

Parameter in IX

Trashes F, HL

Return value in BCDE, carry flag

### Example

```
ld ix,my_pool
lcall pxlast_fast
jr c,.no_elems
; BCDE points to last element
```

### PARAMETERS

**p** Pool handle structure, as previously passed to `pool_xinit()`. Pass this in IX register.

### RETURN VALUE

C flag set: There are no more elements

C flag clear (NC): BCDE points to last element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxlast`, `pxprev_fast`



---

---

## pxnext

---

---

```
long pxnext( Pool_t * p, long e );
```

### DESCRIPTION

Get the next allocated element in an xmem pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

You can easily iterate through all of the allocated elements of a root pool using the following construct:

```
long e;
Pool_t * p;

for (e = pxfirst(p); e; e = pxnext(p, e)) {
    ...
}
```

### PARAMETERS

- p** Pool handle structure, as previously passed to `pool_xinit()`.
- e** Previous element address, obtained by e.g. `pxfirst()`. This must be an allocated element in the given pool, otherwise the results are undefined. Be careful when iterating through a list and deleting elements using `pxfree()`: once the element is deleted, it is no longer valid to pass its address to this function. If this parameter is zero, then the result is the same as `pxfirst()`. This ensures the invariant

$$\text{pxnext}(p, \text{pxprev}(p, e)) == e.$$

### RETURN VALUE

- 0: There are no more elements
- !0: Pointer to the next allocated element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxalloc`, `pxfree`, `pxfirst`, `pxprev`

---

---

## pxnext\_fast

---

---

```
xmem long pxnext_fast( Pool_t * p, long e );
```

### DESCRIPTION

Get the next allocated element in an xmem pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `pxnext()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

### Registers

Parameters in IX, DE respectively

Trashes AF, HL

Return value in BCDE, carry flag

### Example

```
ld ix,my_pool
ld de,(current_element)
ld bc,(current_element+2)
lcall pxnext_fast
jr c,.no_more_elems
; BCDE points to next allocated element
```

### PARAMETERS

- |          |  |
|----------|--|
| <b>p</b> | Pool handle structure, as previously passed to <code>pool_xinit()</code> . Pass this in the IX register. |
| <b>e</b> | Current element, address in BCDE register. See <code>pxnext()</code> for fuller description.             |

### RETURN VALUE

C flag set: There are no more elements

C flag clear (NC): BCDE points to next element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxalloc`, `pxfree`, `pxfirst`, `pxprev`

---

---

## pxprev

---

---

```
long pxprev( Pool_t * p, long e );
```

### DESCRIPTION

Get the previous allocated element in an xmem pool. The pool **MUST** be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise the results are undefined.

You can easily iterate through all of the allocated elements of an xmem pool using the following construct:

```
long e;
Pool_t * p;
for (e = pxlast(p); e; e = pxprev(p, e)) {
    ...
}
```

### PARAMETERS

- |          |   |
|----------|---|
| <b>p</b> | Pool handle structure, as previously passed to <code>pool_xinit()</code> .  |
| <b>e</b> | Previous element address, obtained by e.g., <code>pxlast()</code> . This must be an allocated element in the given pool; otherwise, the results are undefined. Be careful when iterating through a list and deleting elements using <code>pxfree()</code> : once the element is deleted, it is no longer valid to pass its address to this function. If this parameter is zero, then the result is the same as <code>pxlast()</code> . This ensures the invariant |

`pxlast(p, pxnext(p, e)) == e`

### RETURN VALUE

- 0: There are no more elements
- !0: Points to previously allocated element

### LIBRARY

`POOL.LIB`

### SEE ALSO

`pool_xinit`, `pool_link`, `pxalloc`, `pxfree`, `pxlast`, `pxnext`

---

---

## pxprev\_fast

---

---

```
xmem long pxprev_fast( Pool_t * p, long e );
```

### DESCRIPTION

Get the previous allocated element in an xmem pool. The pool MUST be set to being a linked pool using `pool_link(p, <non-zero>)`; otherwise, the results are undefined.

This is an assembler-only version of `pxprev()`.

\*\*\* Do `_not_` call this function from C. \*\*\*

### Registers

Parameters in IX, DE respectively

Trashes AF, HL

Return value in BCDE, carry flag

### Example

```
ld ix,my_pool
ld de,(current_element)
ld bc,(current_element+2)
lcall pxprev_fast
jr c,.no_more_elems
; BCDE points to previously allocated element
```

### PARAMETERS

- |          |  |
|----------|--|
| <b>p</b> | Pool handle structure, as previously passed to <code>pool_xinit()</code> . Pass this in IX register. |
| <b>e</b> | Current element, address in BCDE register. See <code>pxprev()</code> for fuller description.         |

### RETURN VALUE

C flag set: there are no more elements

C flag clear (NC): BCDE points to previous element

### LIBRARY

POOL.LIB

### SEE ALSO

`pool_xinit`, `pool_link`, `pxalloc`, `pxprev`

---

---

## qd\_error

---

---

```
char qd_error( int channel );
```

### DESCRIPTION

Gets the current error bits for that qd channel. This function is intended to be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

**channel**            The channel to read errors from (currently 1 or 2).

### RETURN VALUE

Set of error flags, that can be decoded with the following masks:

QD_OVERFLOW	0x01
QD_UNDERFLOW	0x02

### LIBRARY

QD.LIB (was in R3000.LIB prior to DC 10)

---

---

## qd\_init

---

---

```
void qd_init( int iplevel );
```

### DESCRIPTION

If your board has a Rabbit 3000A microprocessor installed, the quadrature decoder can be set for 10 bit counter operation. For 10 bit operation, add the following macro at the top of your application program.

```
#define QD_10BIT_OPERATION
```

If the above macro is not defined then the quadrature decoder defaults to 8 bit counter operation. With the Rabbit 3000 processor you must use the default 8-bit operation; defining the 10-bit macro will cause a compile time error.

Sample program `Samples/Rabbit3000/QD_Phase_10bit.c` demonstrates the use of the macro.

If your board has a Rabbit 4000 microprocessor installed, the quadrature decoder inputs must be chosen with one of the following defines. Define only one per quadrature decoder.

```
#define QD1_USEPORTD           // use port D pins 1 and 0
#define QD1_USEPORTEL         // use port E pins 1 and 0
#define QD1_USEPORTEH         // use port E pins 5 and 4

#define QD2_USEPORTD           // use port D pins 3 and 2
#define QD2_USEPORTEL         // use port E pins 3 and 2
#define QD2_USEPORTEH         // use port E pins 7 and 6
```

If no macro is defined for a decoder, that decoder will be disabled.

### PARAMETERS

**iplevel**            The interrupt priority for the ISR that handles the count overflow. This should usually be 1.

### LIBRARY

QD.LIB (was in R3000.LIB prior to DC 10)

---

---

## qd\_read

---

---

```
long qd_read( int channel );
```

### DESCRIPTION

Reads the current quadrature decoder count. Since this function waits for a clear reading, it can potentially block if there is enough flutter in the decoder count.

This function is intended to be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

**channel**            The channel to read (currently 1 or 2).

### RETURN VALUE

Returns a signed long for the current count.

### LIBRARY

QD.LIB (was in R3000.LIB prior to DC 10)

---

---

## qd\_zero

---

---

```
void qd_zero( int channel );
```

### DESCRIPTION

Sets the count for a channel to 0. This function is intended to be used with the Rabbit 3000 and Rabbit 4000.

### PARAMETERS

**channel**            The channel to reset (currently 1 or 2)

### LIBRARY

QD.LIB (was in R3000.LIB prior to DC 10)

---

---

## qsort

---

---

```
int qsort( char * base, unsigned n, unsigned s, int (*cmp) () );
```

### DESCRIPTION

Quick sort with center pivot, stack control, and easy-to-change comparison method. This version sorts fixed-length data items. It is ideal for integers, longs, floats and packed string data without delimiters. Raw integers, longs, floats or strings may be sorted, however, the string sort is not efficient.

### PARAMETERS

<b>base</b>	Base address of the raw string data.
<b>n</b>	Number of blocks to sort.
<b>s</b>	Number of bytes in each block.
<b>cmp</b>	User-supplied compare routine for two block pointers, <b>p</b> and <b>q</b> , that returns an int with the same rules used by Unix <code>strcmp (p, q)</code> : = 0: Blocks <b>p</b> and <b>q</b> are equal < 0: <b>p</b> is less than <b>q</b> > 0: <b>p</b> is greater than <b>q</b>  Beware of using ordinary <code>strcmp ()</code> —it requires a null at the end of each string.

### RETURN VALUE

0 if the operation is successful.

### LIBRARY

SYS.LIB

### EXAMPLE - Sorts an array of integers.

```
int mycmp(int *p, int *q){ return (*p - *q);}
const int q[10] = {12,1,3,-2,16,7,9,34,-90,10};
const int p[10] = {12,1,3,-2,16,7,9,34,-90,10};
main() {
    int i;
    qsort(p,10,2,mycmp);
    for(i=0;i<10;++i) printf("%d. %d, %d\n",i,p[i],q[i]);
}
```

Output from the above sample program:

```
0. -90, 12
1. -2, 1
2. 1, 3
3. 3, -2
4. 7, 16
5. 9, 7
6. 10, 9
7. 12, 34
8. 16, -90
9. 34, 10
```



---

---

## rad

---

---

```
float rad( float x );
```

### DESCRIPTION

Convert degrees (360 for one rotation) to radians ( $2\pi$  for a rotation).

### PARAMETERS

**x** Degree value to convert.

### RETURN VALUE

The radians equivalent of degree.

### LIBRARY

SYS.LIB

### SEE ALSO

deg

---

---

## rand

---

---

```
float rand( void );
```

### DESCRIPTION

Returns a uniformly distributed random number in the range  $0.0 \leq v < 1.0$ . Uses algorithm:

$$\text{rand} = ( 5 * \text{rand} ) \text{ modulo } 2^{32}$$

A default seed value is set on startup, but can be changed with the `srand()` function. `rand()` is not reentrant.

### RETURN VALUE

A uniformly distributed random number:  $0.0 \leq v < 1.0$ .

### LIBRARY

MATH.LIB

### SEE ALSO

randb, randg, srand

---

---

## randb

---

---

```
float randb( void );
```

### DESCRIPTION

Uses algorithm:

$$\text{rand} = ( 5 * \text{rand} ) \text{ modulo } 2^{32}$$

A default seed value is set on startup, but can be changed with the `srand()` function. `randb()` is not reentrant.

### RETURN VALUE

Returns a uniformly distributed random number:  $-1.0 \leq v < 1.0$ .

### LIBRARY

`MATH.LIB`

### SEE ALSO

`rand`, `randg`, `srand`

---

---

## randg

---

---

```
float randg( void );
```

### DESCRIPTION

Returns a gaussian-distributed random number in the range  $-16.0 \leq v < 16.0$  with a standard deviation of approximately 2.6. The distribution is made by adding 16 random numbers (see `rand()`). This function is not task reentrant.

### RETURN VALUE

A gaussian distributed random number:  $-16.0 \leq v < 16.0$ .

### LIBRARY

`MATH.LIB`

### SEE ALSO

`rand`, `randb`, `srand`

---

---

## RdPortE

---

---

```
int RdPortE( unsigned int port );
```

### DESCRIPTION

Reads an external I/O register specified by the argument.

### PARAMETERS

**port**                   Address of external parallel port data register.

### RETURN VALUE

Returns an integer, the lower 8 bits of which contain the result of reading the port specified by the argument. Upper byte contains zero.

### LIBRARY

SYSIO.LIB

### SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, BitRdPortE, WrPortE,  
BitWrPortE

---

---

## RdPortI

---

---

```
int RdPortI( int port );
```

### DESCRIPTION

Reads an internal I/O port specified by the argument (use `RdPortE()` for external port).

All of the Rabbit internal registers have predefined macros corresponding to the name of the register. `PADR` is #defined to be `0x30`, etc.

### PARAMETERS

**port**                      Address of internal I/O port

### RETURN VALUE

Returns an integer, the lower 8 bits of which contain the result of reading the port specified by the argument. Upper byte contains zero.

### LIBRARY

`SYSIO.LIB`

### SEE ALSO

`RdPortE`, `BitRdPortI`, `WrPortI`, `BitWrPortI`, `BitRdPortE`, `WrPortE`,  
`BitWrPortE`

---

---

## ReadCompressedFile

---

---

```
int ReadCompressedFile( ZFILE * input, UBYTE * buf, int lenx );
```

### DESCRIPTION

This function decompresses a compressed file (input `ZFILE`, opened with `OpenInputCompressedFile()`) using the LZ compression algorithm on-the-fly, placing a number of bytes (`lenx`) into a user-specified buffer (`buf`).

### PARAMETERS

<code>input</code>	Input bit file.
<code>buf</code>	Output buffer.
<code>lenx</code>	Number of bytes to read. This can be increased to get more throughput or decreased to free up variable space.

### RETURN VALUE

Number of bytes read

### LIBRARY

`LZSS.LIB`

---

---

## read\_rtc

---

---

```
unsigned long read_rtc( void );
```

### DESCRIPTION

Reads seconds (32 bits) directly from the Real-time Clock (RTC). Use with caution! In most cases use long variable `SEC_TIMER`, which contains the same result, unless the RTC has been changed since the start of the program.

If you are running the processor off the 32 kHz crystal and using a Dynamic C version prior to 7.30, use `read_rtc_32kHz()` instead of `read_rtc()`. Starting with DC 7.30, `read_rtc_32kHz()` is deprecated because it is no longer necessary. Programmers should only use `read_rtc()`.

### RETURN VALUE

Time in seconds since January 1, 1980 (if RTC set correctly).

### LIBRARY

`RTCLOCK.LIB`

### SEE ALSO

`write_rtc`

---

---

## read\_rtc\_32kHz

---

---

```
unsigned long read_rtc_32kHz( void );
```

### DESCRIPTION

Reads the real-time clock directly when the Rabbit processor is running off the 32 kHz oscillator. See `read_rtc` for more details.

### RETURN VALUE

Time in seconds since January 1, 1980 (if RTC set correctly).

### LIBRARY

`RTCLOCK.LIB`

---

---

## readUserBlock

---

---

```
int readUserBlock( void * dest, unsigned addr, unsigned numbytes );
```

### DESCRIPTION

Reads a number of bytes from the User block on the primary flash to a buffer in root memory. Please note that portions of the User block may be used by the BIOS for your board to store values. For example, any board with an A to D converter will require the BIOS to write calibration constants to the User block. For some versions of the BL2000 and the BL2100 this memory area is 0x1C00 to 0x1FFF. See the user's manual for your particular board for more information before overwriting any part of the User block. Also, see the *Rabbit Microprocessor Designer's Handbook* for more information on the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), `readUserBlockArray()` should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using  $\mu$ C/OS-II and `_SPI_USE_UCOS_MUTEX` is #defined, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error `ERR_SPI_MUTEX_ERROR` will occur. See the description for `_rcm43_InitUCOSMutex()` for more information on using  $\mu$ C/OS-II and `_SPI_USE_UCOS_MUTEX`.

### PARAMETERS

<b>dest</b>	Pointer to destination to copy data to.
<b>addr</b>	Address offset in User block to read from.
<b>numbytes</b>	Number of bytes to copy.

### RETURN VALUE

- 0: Success
- 1: Invalid address or range
- 2: No valid ID block found (block version 3 or later)

The return values below are applicable only if `_SPI_USE_UCOS_MUTEX` is not #defined:

- ETIME: (Serial flash only, time out waiting for SPI)
- postive N: (Serial flash only, SPI in use by device N)

### LIBRARY

IDBLOCK.LIB

### SEE ALSO

`writeUserBlock`, `readUserBlockArray`

---

---

## readUserBlockArray

---

---

```
int readUserBlockArray( void * dests[], unsigned numbytes[], int
    numdests, unsigned addr );
```

### DESCRIPTION

Reads a number of bytes from the User block on the primary flash to a set of buffers in root memory. This function is usually used as the inverse function of `writeUserBlockArray()`.

This function was introduced in Dynamic C version 7.30.

**Note:** Portions of the User block may be used by the BIOS to store values such as calibration constants. See the manual for your particular board for more information before overwriting any part of the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), `readUserBlockArray()` should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using  $\mu$ C/OS-II and `_SPI_USE_UCOS_MUTEX` is `#defined`, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error `ERR_SPI_MUTEX_ERROR` will occur. See the description for `_rcm43_InitUCOSMutex()` for more information on using  $\mu$ C/OS-II and `_SPI_USE_UCOS_MUTEX`.

### PARAMETERS

<b>dests</b>	Pointer to array of destinations to copy data to.
<b>numbytes</b>	Array of numbers of bytes to be written to each destination.
<b>numdests</b>	Number of destinations.
<b>addr</b>	Address offset in User block to read from.

### RETURN VALUE

0: Success  
-1: Invalid address or range  
-2: No valid System ID block found (block version 3 or later)  
The return values below are applicable only if `_SPI_USE_UCOS_MUTEX` is not `#defined`:  
-ETIME: (Serial flash only, time out waiting for SPI)  
postive N: (Serial flash only, SPI in use by device N)

### LIBRARY

IDBLOCK.LIB

### SEE ALSO

`writeUserBlockArray`, `readUserBlock`



---

---

## res

---

---

```
void res( void * address, unsigned int bit );
```

### DESCRIPTION

Dynamic C may expand this call inline. Clears specified bit at memory address to 0. Bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address &= ~(1L << bit)
```

### PARAMETERS

**address**            Address of byte containing bits 7-0.  
**bit**                 Bit location where 0 represents the least significant bit.

### LIBRARY

UTIL.LIB

### SEE ALSO

RES

---

---

## RES

---

---

```
void RES( void * address, unsigned int bit );
```

### DESCRIPTION

Dynamic C may expand this call inline. Clears specified bit at memory address to 0. `bit` may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address &= ~(1L << bit)
```

### PARAMETERS

**address**            Address of byte containing bits 7-0.  
**bit**                 Bit location where 0 represents the least significant bit.

### LIBRARY

UTIL.LIB

### SEE ALSO

res

---

---

## ResetErrorLog

---

---

```
void ResetErrorLog( void );
```

### DESCRIPTION

This function resets the exception and restart type counts in the error log buffer header. This function is not called by default from anywhere. It should be used to initialize the error log when a board is programmed by means other than Dynamic C, cloning, the Rabbit Field Utility (RFU), or a service processor. For example, if boards are mass produced with pre-programmed flash chips, then the test program that runs on the boards should call this function.

### LIBRARY

ERRORS.LIB

---

---

## root2vram

---

---

```
int root2vram( void * src, int start, int length );
```

### DESCRIPTION

This function copies data to the VBAT RAM. Tamper detection on the Rabbit 4000 erases the VBAT RAM with any attempt to enter bootstrap mode.

### PARAMETERS

<b>src</b>	The address to the data in root to be copied to vbat ram.
<b>start</b>	The start location within the VBAT RAM (0-31).
<b>length</b>	The length of data to write to VBAT RAM. The length should be greater than 0. The parameters <code>length + start</code> should not exceed 32.

### LIBRARY

VBAT.LIB

### SEE ALSO

vram2root

---

---

## root2xmem

---

---

```
int root2xmem( unsigned long dest, void * src, unsigned len );
```

### DESCRIPTION

Stores `len` characters from logical address `src` to physical address `dest`.

### PARAMETERS

<code>dest</code>	Physical address.
<code>src</code>	Logical address.
<code>len</code>	Numbers of bytes.

### RETURN VALUE

0: Success.  
-1: Attempt to write flash memory area, nothing written.  
-2: Source not all in root.

### LIBRARY

XMEM.LIB

### SEE ALSO

`xalloc`, `xmem2root`

---

---

## rtc\_timezone

---

---

```
int rtc_timezone( long * seconds, char * tzname );
```

### DESCRIPTION

This function returns the timezone offset as known by the library. The timezone is obtained from the following sources, in order of preference:

1. The DHCP server. This can only be used if the TCP/IP stack is in use, and `USE_DHCP` is defined.
2. The `TIMEZONE` macro. This should be defined by the program to an `_hour_` offset - may be floating point.

### PARAMETERS

<b>seconds</b>	Pointer to result longword. This will be set to the number of seconds offset from Coordinated Universal Time (UTC). The value will be negative for west; positive for east of Greenwich.
<b>tzname</b>	If null, no timezone name is returned. Otherwise, this must point to a buffer of at least 7 bytes. The buffer is set to a null-terminated string of between 0 and 6 characters in length, according to the value of the <code>TZNAME</code> macro. If <code>TZNAME</code> is not defined, then the returned string is zero length ("").

### RETURN VALUE

- 0: timezone obtained from DHCP.
- 1: timezone obtained from `TIMEZONE` macro. The value of this macro (which may be `int`, `float` or a variable name) is multiplied by 3600 to form the return value.
- 2: timezone is zero since the `TIMEZONE` macro was not defined.

### LIBRARY

`RTCLOCK.LIB`

---

---

## runwatch

---

---

```
void runwatch( void );
```

### DESCRIPTION

Runs and updates watch expressions if Dynamic C has requested it with a Ctrl-U. Should be called periodically in user program.

### LIBRARY

SYS.LIB

---

---

## sdspi\_debounce

---

---

```
int sdspi_debounce( sd_device * sd );
```

### DESCRIPTION

This function waits for and debounces the card insertion switch. When it returns True (1), then a card is fully inserted.

### PARAMETER

**sd**                      The device structure for the SD card.

### RETURN VALUE

1: Success, card fully inserted  
0: No card present

### LIBRARY

SDFLASH.LIB

---

---

## `sdspi_get_csd`

---

---

```
int sdspi_get_csd( sd_device * sd );
```

### DESCRIPTION

This function is called to execute protocol command 9 to retrieve the SD card's Card Specific Data (CSD) and store it in the respective SD driver configuration object. The CSD data is used to determine the SD card's physical storage and timing attributes.

### PARAMETERS

`sd`                    The device structure for the SD card.

### RETURN VALUE

0: Success  
- EIO: I/O error  
- EINVAL: Invalid parameter given  
- ENOMEDIUM: No SD card in socket  
- ESHAREDDBUSY: Shared SPI port busy

### LIBRARY

SDFLASH.LIB

---

---

## `sdspi_get_scr`

---

---

```
int sdspi_get_scr( sd_device * sd );
```

### DESCRIPTION

This function executes application specific command 51 to retrieve the SD card's Configuration Register (SCR) and store it in the respective SD driver configuration object. The SCR data is used to identify the SD card's physical interface version and security version. It also contains erase state (all 0's or 1's) and supported bus widths.

### PARAMETERS

**sd**                      The device structure for the SD card.

### RETURN VALUE

0: Success  
-EIO: I/O error  
-EINVAL: Invalid parameter given  
-ENOMEDIUM: No SD card in socket  
-ESHAREDBUSY: Shared SPI port busy

### LIBRARY

`SDFLASH.LIB`

---

---

## sdspi\_getSectorCount

---

---

```
long sdspi_getSectorCount( sd_device * dev );
```

### DESCRIPTION

Return number of usable 512 byte sectors on an SD card.

### PARAMETER

**dev** Pointer to `sd_device` struct for initialized flash device.

### RETURN VALUE

Number of sectors

### LIBRARY

SDFLASH.LIB

---

---

## sdspi\_get\_status\_reg

---

---

```
int sdspi_get_status_reg( sd_device *sd, int * status );
```

### DESCRIPTION

This function is called to execute protocol command 13 to retrieve the status register value of the SD card.

### PARAMETERS

**sd** Pointer to the device structure for the SD card.

**status** Pointer to variable that returns the status.

### RETURN VALUE

0: Success, Card status placed in status  
- EIO: I/O error  
- ENOMEDIUM: No SD card in socket  
- ESHAREDBUSY: Shared SPI port busy

### LIBRARY

SDFLASH.LIB



---

---

## `sdspi_init_card`

---

---

```
int sdsppi_init_card( sd_device * sd );
```

### DESCRIPTION

Initializes the SD card pointed to by `sd`. Function executes protocol command “1” which clears HCS bit and activates the card’s initialization sequence.

### PARAMETERS

**`sd`**                      Pointer to `sd_device` structure for the SD card.

### RETURN VALUE

0: Success  
- EIO: I/O error  
- EINVAL: Invalid parameter given  
- ENOMEDIUM: No SD card in socket  
- ESHARED BUSY: Shared SPI port busy

### LIBRARY

`SDFLASH.LIB`

---

---

## **sdspi\_initDevice**

---

---

```
int sdspi_initDevice( int indx, sd_dev_interface * sd_dev );
```

### **DESCRIPTION**

Initializes the SD card pointed to by `sd_dev` and adds information about the cards interface to the SD device array in the position pointed to by `indx`. Sets up the default block size of 512 bytes used by sector read/write functions. This function should be called before any calls to other `sdspi` functions.

### **PARAMETERS**

<b>indx</b>	Index into the SD device array to add the card.
<b>sd_dev</b>	Pointer to <code>sd_dev_interface</code> for the SD card.

### **RETURN VALUE**

0: Success  
-EIO: I/O error  
-EINVAL: Invalid parameter given  
-ENOMEDIUM: No SD card in socket  
-ESHAREDBUSY: SPI port busy

### **LIBRARY**

SDFLASH.LIB

---

---

## sdspi\_isWriting

---

---

```
int sdspi_isWriting( sd_device * dev );
```

### DESCRIPTION

Returns 1 if the SD card is busy writing a sector.

### PARAMETER

**dev**                      Pointer to initialized sd\_device structure for the flash chip

### RETURN VALUE

1: Busy  
0: Ready, not currently writing

### LIBRARY

SDFLASH.LIB

---

---

## sdspi\_notbusy

---

---

```
int sdspi_notbusy( int port );
```

### DESCRIPTION

This function tests for a busy status from the SD card on the port given. It is assumed that the card is already enabled.

### PARAMETER

**port**                      The base address for the SD card's SPI port

### RETURN VALUE

1: The card is not busy, write/erase has ended  
0: The card is busy, write/erase in progress

### LIBRARY

SDFLASH.LIB

---

---

## `sdspi_print_dev`

---

---

```
void sdspi_print_dev( sd_device * dev );
```

### DESCRIPTION

Prints parameters from the SD device structure.

### PARAMETER

**dev**                    Pointer to `sd_device` structure of the SD card.

### LIBRARY

SDFLASH.LIB

---

---

## **sdspi\_process\_command**

---

---

```
int sdspi_process_command( sd_device *sd, SD_CMD_REPLY * cmd_reply,
    int mode );
```

### **DESCRIPTION**

This function sends the command placed in the `cmd_reply` structure and retrieves a reply and data (optional) as defined in the `cmd_reply` structure. Pointers to TX and RX buffers are retrieved from the `cmd_reply` structure and used for command transmission and reply/data reception. Reply is parsed and placed in `cmd_reply.reply`. Errors encountered will give a negative return value.

The SPI semaphore is obtained before the command is sent. The mode parameter controls whether the semaphore will be released after command execution and reply/data reception. If mode is zero, both semaphore and chip select are active on a successful return. An end command sequence and release of the semaphore must be handled by caller.

If mode is not 0, the semaphore will be released before returning. In addition, if mode is 2 then an SD card reset is in progress. This enables the distinguishing of certain I/O error conditions that would normally be grouped with the `-EIO` error code and instead return the `-EAGAIN` error code, indicating reset retries should continue.

### **PARAMETER**

<b>sd</b>	Pointer to <code>sd_device</code> structure of the SD card.
<b>cmd_reply</b>	Pointer to <code>cmd_reply</code> structure, which contains: <code>cmd</code> - command to be executed <code>argument</code> - arguments for the command <code>reply</code> - storage for command reply <code>reply_size</code> - size in bytes of expected reply <code>data_size</code> - size in bytes of expected data <code>tx_buffer</code> - pointer to TX buffer to use <code>rx_buffer</code> - pointer to RX buffer to use
<b>mode</b>	One of the following: 0 = SPI port semaphore should be retained. 1 = If SPI port to be released before return. 2 = Attempting SD card reset, otherwise same as mode "1". (Enables <code>-EAGAIN</code> return value.)

---

---

## sdspi\_process\_command (cont'd)

---

---

### RETURN VALUE

- 0: Success
- EIO: I/O error
- EAGAIN: Allowable I/O error during card reset
- EINVAL: Invalid parameter given
- ENOMEDIUM: No SD card in socket
- ESHAREDBUSY: Shared SPI port busy

### LIBRARY

SDFLASH.LIB

---

---

## sdspi\_read\_sector

---

---

```
int sdspi_read_sector( sd_device * sd, unsigned long sector_number,
    void * data_buffer );
```

### DESCRIPTION

This function is called to execute protocol command 17 to read a 512 byte block of data from the SD card.

### PARAMETER

- |                      |   |
|----------------------|---|
| <b>sd</b>            | Pointer to <code>sd_device</code> structure of the SD card. |
| <b>sector_number</b> | The sector number to read.                                  |
| <b>data_buffer</b>   | Pointer to a buffer for the 512 bytes read.                 |

### RETURN VALUE

- 0: Success
- EIO: I/O error
- EINVAL: Invalid parameter given
- ENOMEDIUM: No SD card in socket
- ESHAREDBUSY: Shared SPI port busy

### LIBRARY

SDFLASH.LIB

---

---

## `sdspi_reset_card`

---

---

```
int sdspi_reset_card( sd_device * sd );
```

### DESCRIPTION

Resets the SD card pointed to by `sd`. Function executes protocol command 0 to force the card to Idle mode. This command is sent multiple times to reset the SD card.

### PARAMETER

`sd`                      Pointer to `sd_device` structure of the SD card.

### RETURN VALUE

0: Success  
- EIO: I/O error  
- EINVAL: Invalid parameter given  
- ENOMEDIUM: No SD card in socket  
- ESHAREDBUSY: Shared SPI port busy

### LIBRARY

`SDFLASH.LIB`

---

---

## sdspi\_sendingAP

---

---

```
int sdspi_sendingAP( sd_device * sd );
```

### DESCRIPTION

Sends AP command 55 to set Alternate Command mode on the next command sent to the card. This function does not release the port sharing semaphore unless an error is encountered.

### PARAMETER

**sd** Pointer to `sd_device` structure of the SD card.

### RETURN VALUE

0: Success  
- EIO: I/O error  
- ENOMEDIUM: No SD card in socket  
- ESHAREDBUSY: Shared SPI port busy

### LIBRARY

SDFLASH.LIB

---

---

## sdspi\_setLED

---

---

```
void sdspi_setLED( sd_device * sd, char state );
```

### DESCRIPTION

This function sets the LED for the given SD card based on state. If state is 0, the LED is turned off. If state is not zero, the LED is turned on.

### PARAMETER

**sd** Pointer to `sd_device` structure of the SD card.  
**state** The state to set the LED to: 0 = Off and Non-zero = On

### LIBRARY

SDFLASH.LIB



---

---

## `sdspi_set_block_length`

---

---

```
int sdsapi_set_block_length( sd_device * sd, int block_length );
```

### DESCRIPTION

This function executes protocol command 16 to set the block length for the SD card. The default block length for SD cards is 512 bytes. Please note that `sdsapi_write_sector()` and `sdsapi_read_sector()` work on 512 byte blocks only. If you change the block size, these functions will need to be modified, or you will need to execute commands directly through `sdsapi_process_command()` and internal write block and read block functions.

### PARAMETER

<code>sd</code>	Pointer to device structure of the SD card.
<code>block_length</code>	The block size in bytes for the SD card.

### RETURN VALUE

0: Success  
-EIO: I/O error  
-EINVAL: Invalid parameter given  
-ENOMEDIUM: No SD card in socket  
-ESHAREDBUSY: Shared SPI port busy

### LIBRARY

`SDFLASH.LIB`

---

---

## sdspi\_WriteContinue

---

---

```
int sdspi_WriteContinue( sd_device * sd );
```

### DESCRIPTION

This function completes the previously started write command to the SD card when non-blocking mode is enabled. It looks for the end of the busy signal from the card, then strobes the chip select. This function should be called repeatedly until the -EBUSY code is not returned, at which point the SPI port is freed. There is a timeout mechanism for the busy signal. If exceeded, the port is freed and the -EIO error code is returned.

### PARAMETERS

**sd**                    The device structure for the SD card.

### RETURN VALUE

0: Success  
-EIO: I/O error or timeout  
-EBUSY: SD card is busy with write operation; call `sdspi_WriteContinue()` again

### LIBRARY

SDFLASH.LIB

---

---

## **sdspi\_write\_sector**

---

---

```
int sdspi_write_sector( sd_device * sd, unsigned long sector_number,
    char * data_buffer );
```

### **DESCRIPTION**

This function is called to execute protocol command 24 to write a 512 byte block of data to the SD card.

### **PARAMETER**

<b>sd</b>	Pointer to device structure of the SD card.
<b>sector_number</b>	The sector number to write.
<b>data_buffer</b>	Pointer to a buffer of 512 bytes to write.

### **RETURN VALUE**

0: Success  
-EIO: I/O error  
-EACCES: Write protected block, no write access  
-EINVAL: Invalid parameter given  
-ENOMEDIUM: No SD card in socket  
-ESHAREDBUSY: Shared SPI port busy  
-EBUSY: SD card is busy with write operation; call `sdspi_WriteContinue()` to complete (only when `SD_NON_BLOCK` is defined)

### **LIBRARY**

SDFLASH.LIB

---

---

## servo\_alloc\_table

---

---

```
void servo_alloc_table( int which, int entries );
```

### DESCRIPTION

Allocate an xmem data area for servo statistics collection. This function should be called once only (for each servo) at application startup time.

### PARAMETERS

<b>which</b>	Servo (0 or 1)
<b>entries</b>	Number of entries to allocate. Each entry is 8 bytes, and stores 4 integer values. The maximum value for this parameter is 8190.

### LIBRARY

SERVO.LIB

### SEE ALSO

servo\_graph, servo\_read\_table, servo\_stats\_reset

---

---

## servo\_closedloop

---

---

```
void servo_closedloop( int which, int reset );
```

### DESCRIPTION

Run specified servo in closed-loop (PID) mode.

### PARAMETERS

<b>which</b>	Servo (0 or 1).
<b>reset</b>	Whether to reset the current command list. The command list executes even while in open loop mode (although it will have no visible effect in that mode). If reset is non-zero, then the command list will be reset to empty and the motor will halt at the current position.

### LIBRARY

SERVO.LIB

### SEE ALSO

servo\_openloop, servo\_torque

---

---

## servo\_disable\_0

---

---

```
void servo_disable_0( void );
```

### DESCRIPTION

Disable drive to the first servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```
#define SERVO_ENABLE_PORT_0          PGDR
#define SERVO_ENABLE_PORTSHADOW_0   PGDRShadow
#define SERVO_ENABLE_PIN_0          6
```

and, optionally,

```
#define SERVO_ENABLE_DDR_0          PGDDR
#define SERVO_ENABLE_DDRSHADOW_0   PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_0
```

This function is limited to toggling the output pin. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```
#define SERVO_DISABLE_0  yyy
```

where yyy is the name of your own function (which is assumed to take no parameters and have no return value)

### LIBRARY

```
SERVO.LIB
```

### SEE ALSO

```
servo_enable_0
```

---

---

## servo\_disable\_1

---

---

```
void servo_disable_1( void );
```

### DESCRIPTION

Disable drive to the second servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```
#define SERVO_ENABLE_PORT_1          PGDR
#define SERVO_ENABLE_PORTSHADOW_1   PGDRShadow
#define SERVO_ENABLE_PIN_1          7
```

and, optionally,

```
#define SERVO_ENABLE_DDR_1          PGDDR
#define SERVO_ENABLE_DDRSHADOW_1   PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_1
```

This function is limited to toggling the output pin. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```
#define SERVO_DISABLE_1  yyy
```

where yyy is the name of your own function (which is assumed to take no parameters and have no return value)

### LIBRARY

```
SERVO.LIB
```

### SEE ALSO

```
servo_enable_1
```

---

---

## servo\_enable\_0

---

---

```
void servo_enable_0( void );
```

### DESCRIPTION

Enable drive to the first servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```
#define SERVO_ENABLE_PORT_0 PGDR
#define SERVO_ENABLE_PORTSHADOW_0 PGDRShadow
#define SERVO_ENABLE_PIN_0 6
```

and, optionally,

```
#define SERVO_ENABLE_DDR_0 PGDDR
#define SERVO_ENABLE_DDRSHADOW_0 PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_0
```

This function is limited to toggling the output pin high or low. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```
#define SERVO_ENABLE_0 xxxx
```

where xxxx is the name of your own function (which is assumed to take no parameters and have no return value).

### LIBRARY

```
SERVO.LIB
```

### SEE ALSO

```
servo_disable_0
```

---

---

## servo\_enable\_1

---

---

```
void servo_enable_1( void );
```

### DESCRIPTION

Enable drive to the second servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```
#define SERVO_ENABLE_PORT_1 PGDR
#define SERVO_ENABLE_PORTSHADOW_1 PGDRShadow
#define SERVO_ENABLE_PIN_1 7
```

and, optionally,

```
#define SERVO_ENABLE_DDR_1 PGDDR
#define SERVO_ENABLE_DDRSHADOW_1 PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_1
```

This function is limited to toggling the output pin high or low. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```
#define SERVO_ENABLE_1 xxxx
```

where xxxx is the name of your own function (which is assumed to take no parameters and have no return value).

### LIBRARY

```
SERVO.LIB
```

### SEE ALSO

```
servo_disable_1
```



---

---

## servo\_gear

---

---

```
void servo_gear( int count0, int count1, int slave0, int slave1 );
```

### DESCRIPTION

**NOTE:** this function is currently not efficient enough for production use (owing to use of long multiplication and division). It is provided as an example of the use of callbacks from the ISR.

If two servos are in use, couple or cross-couple their positioning. This only works if NUM\_SERVOS is 2, and both servos are in closed loop mode.

There are four possible sub-modes of operation, which depend on the slave0/1 parameters.

slave0	slave1	Operation
0	0	Non-gear mode: neither servo is slaved. This is the normal, default, mode.
0	1	Second servo is slaved from first servo. For every 'count0' increments of the first servo's encoder, the second servo will be moved 'count1' increments.
1	0	First servo is slaved from second servo. For every 'count1' increments of the second servo's encoder, the first servo will be moved 'count0' increments.
1	1	Both servos cross-coupled. Movement will only result from an externally applied torque. This is a true simulation of mechanical gearing.

Call this function with count0 or count1 zero, or both slave0 and slave1 zero, to exit from gear mode. When a servo that was slaved is set to normal mode, its velocity is set to zero.

### PARAMETERS

**count0** Encoder increment for the first servo which results from count1 increments of the second servo.

**count1** Encoder increment for the second servo which results from count0 increments of the first servo.

Together, count0 and count1 determine the gearing ratio. Neither value should be set to a magnitude greater than about 500, to avoid internal arithmetic overflow. In any gear mode, the total movement of either servo should be limited to less than about 2M counts in either direction from the point at which gear mode was set. If a smaller range of movement is acceptable, then the maximum of either count parameter may be increased proportionally. The value of count0/count1 or count1/count0 should not have a magnitude greater than about 10 to avoid encoder quantization problems, especially in cross-coupled mode.

---

---

## `servo_gear (cont'd)`

---

---

`slave0`            1 if first servo slaved to second, else zero.

`slave1`            1 if second servo slaved to first, else zero.

### **LIBRARY**

`SERVO.LIB`

### **SEE ALSO**

`servo_closedloop`, `servo_torque`

---

---

## servo\_graph

---

---

```
int servo_graph( int which, word start, word nlines, word samples,  
                word what, int low, int high );
```

### DESCRIPTION

Draw ASCII-art graph of servo response. This is primarily intended for debugging. It should be called after resetting the sample collection table using `servo_stats_reset()`, then executing a movement whose response is to be graphed.

### PARAMETERS

<b>which</b>	Servo (0 or 1)
<b>start</b>	Starting sample number
<b>nlines</b>	Number of lines (sample bins) in graph - vertical axis
<b>samples</b>	Number of samples to cover (should be multiple of nlines)
<b>what</b>	Which statistic to print: 0 is for error; 1 for error integral; 2 for error rate (differential), 3 for PWM output setting. These may be customized to have different meanings
<b>low</b>	Low range of horizontal axis
<b>high</b>	High range of horizontal axis

### RETURN VALUE

0: OK  
-1: error

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_alloc_table`, `servo_read_table`, `servo_stats_reset`

---

---

## servo\_init

---

---

```
void servo_init( void );
```

### DESCRIPTION

This function must be called once at the beginning of application code to initialize the servo library.

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_stats_reset`, `servo_alloc_table`, `servo_set_coeffs`,  
`servo_enable_0`

---

---

## servo\_millirpm2vcmd

---

---

```
long servo_millirpm2vcmd( int which, long millirpm );
```

### DESCRIPTION

Convert 1/1000 RPM units to velocity command value. Basic formula is:

$$\text{vcmd} = \frac{\text{SERVO\_COUNT\_PER\_REV\_n} \cdot \text{millirpm} \cdot 65536}{60000 \cdot \text{SERVO\_LOOP\_RATE\_HZ}}$$

Floating point is used to retain 24 bit precision.

### PARAMETERS

<b>which</b>	Servo (0 or 1).
<b>millirpm</b>	Input in units of 1/1000 RPM.

### RETURN VALUE

Output in units suitable for command velocity setting i.e units of 1/65536 encoder counts per ISR execution (sample).

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_move_to`, `servo_set_vel`, `servo_set_pos`

---

---

## servo\_move\_to

---

---

```
int servo_move_to( int which, long pos, long ticks, long accel_ticks,
                  long final_v );
```

### DESCRIPTION

Move to new position, pos. Assumes current position is "cmd" and current velocity is "vcmd" (with the values of these read from the control structure at beginning of routine).

Each "tick" represents the time interval between loop updates. This routine measures time intervals in units of ticks.

accel\_ticks ( $\leq$  ticks) is the number of ticks allocated to acceleration/deceleration phase of movement. The remaining part of the movement is performed at constant velocity. Acceleration and deceleration are computed to be of the same magnitude at beginning and end of motion (but may be opposite signs). final\_v is the velocity to be achieved at end of movement. This routine returns as soon as the necessary command list is installed for execution by the ISR. The movement will not be completed until "ticks" ISR executions.

NB: if the average velocity (vt) required to complete the movement is greater than  $\pm 16k$  counts per tick, then the movement is stretched to a longer time interval so as to make the peak velocity equal to the  $\pm 8k$  counts/tick (which is higher than any physical motor can follow). accel\_ticks is set to 16384 if it is over that (since rounding errors can accumulate over long periods of low acceleration).

If this routine is called again before the previous motion is completed, then the previous motion will be overridden by the new motion. This routine uses floating point, since the mathematics are quite complex. It takes several milliseconds to execute, so should not be called to perform motions which complete in less than, say, 50ms.

This routine does not attempt to control rate of change of acceleration ("jerk" or  $d^3x/dt^3$ ). It approximates the required movement profile as parabolic (constant acceleration) and linear (constant velocity) segments.

### PARAMETERS

<b>which</b>	Servo (0 or 1).
<b>pos</b>	Position to be achieved at end of movement.
<b>ticks</b>	Number of ISR executions (loop update rate) over which to complete the movement. If less than 1, it is set to 1.
<b>accel_ticks</b>	Number of ticks over which acceleration is to be applied. The remainder of the interval, ticks - accel_ticks, is performed at constant velocity. If greater than "ticks", it is set equal to "ticks".
<b>final_v</b>	Final velocity to be achieved at end of movement.

---

---

## `servo_move_to (cont'd)`

---

---

### RETURN VALUE

0: OK.

1: computed velocity is "extremely high": time interval stretched to make velocity fit within allowable fixed-point limits (i.e. 8192 encoder counts per sample).

### LIBRARY

`SERVO.LIB`

### SEE ALSO

`servo_set_vel`, `servo_set_pos`, `servo_millirpm2vcmd`

---

---

## `servo_openloop`

---

---

```
void servo_openloop( int which, word pwm );
```

### DESCRIPTION

Run specified servo in open-loop mode (no PID control). Note that this bypasses dynamic current-limiting (if any defined) so should be used with caution.

### PARAMETERS

**which**            Servo (0 or 1).

**pwm**              Output PWM setting (0-1024). 0 indicates maximum reverse speed, 1024 is maximum forward speed. 512 is nominally zero speed (but this depends on amplifier offset).

### LIBRARY

`SERVO.LIB`

### SEE ALSO

`servo_closedloop`, `servo_torque`

---

---

## servo\_qd\_zero\_0

---

---

```
void servo_qd_zero_0( void );
```

### DESCRIPTION

Reset the first servo encoder reading to zero. The servo motor is not moved; only the notion of the current position is reset to zero. This should only be called when the servo is in open loop mode.

### LIBRARY

SERVO.LIB

### SEE ALSO

servo\_qd\_zero\_1

---

---

## servo\_qd\_zero\_1

---

---

```
void servo_qd_zero_1 (void ;)
```

### DESCRIPTION

Reset the second servo encoder reading to zero. The servo motor is not moved; only the notion of the current position is reset to zero. This should only be called when the servo is in open loop mode.

### LIBRARY

SERVO.LIB

### SEE ALSO

servo\_qd\_zero\_0

---

---

## servo\_read\_table

---

---

```
int servo_read_table(int which, word entry, word nent, int data[12]);
```

### DESCRIPTION

Read one or more table entries, returning average, max and min of all samples in the specified group starting at entry, for nent samples.

### PARAMETERS

<b>which</b>	Servo (0 or 1)
<b>entry</b>	First sample number
<b>nent</b>	Number of entries starting at "entry"
<b>data [12]</b>	Returned data: 3 sets of 4 contiguous entries. The first set (data[0]..data[3]) contains the average; the second set (data[4]..data[7]) contains the maximum; and the last set (data[8]..data[11]) contains the minimum. The elements of each set correspond with the table data: the first element is the instantaneous error; the second is the error integral; the third is the error rate; and the 4th is the PWM output. These may be customized to have different meanings.

### RETURN VALUE

- 0: OK
- 1: no such entry or entries.

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_alloc_table`, `servo_graph`, `servo_stats_reset`



---

---

## `servo_set_coeffs`

---

---

```
void servo_set_coeffs( int which, int prop, int integral, int diff );
```

### DESCRIPTION

Set the PID closed loop control coefficients. The normal sign for all coefficients should be positive in order to implement a stable control loop. See Technical Note 233 for details.

### PARAMETERS

<code>which</code>	Servo (0 or 1)
<code>prop</code>	Proportional coefficient
<code>integral</code>	Integral ("reset") coefficient
<code>diff</code>	Derivative ("rate") coefficient

### LIBRARY

`SERVO.LIB`

### SEE ALSO

`servo_closedloop`, `servo_openloop`

---

---

## `servo_set_pos`

---

---

```
void servo_set_pos( int which, long pos, long vel );
```

### DESCRIPTION

Move the specified servo motor to a specified position and set the specified velocity at that position. This cancels any move which is currently in effect.

### PARAMETERS

<code>which</code>	Servo (0 or 1)
<code>pos</code>	Position, as an encoder count
<code>vel</code>	Velocity, in units of encoder counts per loop update interval, times 65536. You can convert RPM to a suitable velocity command using <code>servo_millirpm2vcmd()</code> .

### LIBRARY

`SERVO.LIB`

### SEE ALSO

`servo_move_to`, `servo_set_vel`, `servo_millirpm2vcmd`

---

---

## servo\_set\_vel

---

---

```
void servo_set_vel( int which, long vel );
```

### DESCRIPTION

Move the specified servo motor at a constant velocity. This cancels any move that is currently in effect.

### PARAMETERS

<b>which</b>	Servo (0 or 1).
<b>vel</b>	Velocity, in units of encoder counts per loop update interval, times 65536. You can convert RPM to a suitable velocity command using <code>servo_millirpm2vcmd()</code> .

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_move_to`, `servo_set_pos`, `servo_millirpm2vcmd`

---

---

## servo\_stats\_reset

---

---

```
void servo_stats_reset( int which );
```

### DESCRIPTION

Reset the statistics table. This is used immediately prior to a command movement, so that the table is filled with the results of the movement command. Once reset, one table row is filled in for each execution of the update loop (ISR driven). This continues until the table is full, or it is reset again.

### PARAMETER

<b>which</b>	Servo (0 or 1)
--------------	----------------

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_graph`, `servo_read_table`

---

---

## servo\_torque

---

---

```
void servo_torque( int which, int torque );
```

### DESCRIPTION

Run specified servo in open loop controlled torque mode. The torque is limited by the dynamic current limit feature, if available.

### PARAMETERS

<b>which</b>	Servo (0 or 1)
<b>torque</b>	Amount of torque expressed as a fraction of the maximum permissible torque, times 10,000. For example, to set the torque to 1/10 the maximum value in the reverse direction, call <code>servo_torque(0, -1000)</code> .

### LIBRARY

SERVO.LIB

### SEE ALSO

`servo_closedloop`, `servo_openloop`

---

---

## **serCheckParity**

---

---

```
int serCheckParity( char rx_byte, char parity );
```

### **DESCRIPTION**

This function is different from the other serial routines in that it does not specify a particular serial port. This function takes any 8-bit character and tests it for correct parity. It will return true if the parity of `rx_byte` matches the parity specified. This function is useful for checking individual characters when using a 7-bit data protocol.

### **PARAMETERS**

<b>rx_byte</b>	The 8 bit character being tested for parity.
<b>parity</b>	The character 'O' for odd parity, or the character 'E' for even parity.

### **RETURN VALUE**

1: Parity of the byte being tested matches the parity supplied as an argument.  
0: Parity of the byte does not match.

### **LIBRARY**

RS232.LIB

---

---

## serXclose

---

---

```
void serXclose(); /* where X is A-F */
```

### DESCRIPTION

Disables serial port X. This function is non-reentrant.

The functions `serEclose()` and `serFclose()` may be used with the Rabbit 3000 and Rabbit 4000.

### LIBRARY

RS232.LIB

---

---

## serXdatabits

---

---

```
void serXdatabits ( state ); /* where X is A-F */
```

### DESCRIPTION

Sets the number of data bits in the serial format for this channel. Currently seven or eight bit modes are supported. A call to `serXopen()` must be made before calling this function. This function is non-reentrant.

The functions `serEdatabits()` and `serFdatabits()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXdatabits(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

**state**                    An integer indicating what bit mode to use. It is best to use one of the macros provided for this:

- `PARAM_7BIT` - Configures serial port to use 7 bit data.
- `PARAM_8BIT` - Configures serial port to use 8 bit data (default condition).

### LIBRARY

RS232.LIB

---

---

## serXdmaOff

---

---

```
int serXdmaOff( void ); /* where X is A-F */
```

### DESCRIPTION

Stops DMA transfers and unallocates the channels. Restarts the serial interrupt capability.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the function prototype is: `serXdmaOff(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

0: Success  
DMA Error codes: Error

### LIBRARY

RS232.LIB

### SEE ALSO

`serXdmaOn`

---

---

## serXdmaOn

---

---

```
int serXdmaOn( int tcmask, int rcmask ); /* where X is A-F */
```

### DESCRIPTION

Enables DMA for serial send and receive. This function should be called directly after `serXopen()`.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the function prototype is: `serXdmaOn(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

#### Important Flow Control Note:

Because the DMA flowcontrol uses the external request feature, only two serial ports can use DMA flowcontrol at a time. For the CTS pin, one serial port can use PD2, PE2, or PE6, and the other can use PD3, PE3 or PE7.

#### How DMA Serial Works:

DMA Transmit:

When a serial function is called to transmit data, a DMA transfer begins. The length of that transfer is either the length requested, or the rest of the transmit buffer size from the current position. An interrupt is fired at the end of the transmit at which time another transmit is set up if more data is ready to go.

DMA Receive:

When `serXdmaOn()` is called, a continuous chain of DMA transfers begins sending any data received on the serial line to the circular buffer. With flowcontrol on, there is an interrupt after each segment of the data transfer. At that point, if receiving another segment would overwrite data, the `RTSoff` function is called.

For more information see the description at the beginning of `RS232.LIB`.

### PARAMETERS

<b>tcmask</b>	Channel mask for DMA transmit. Use <code>DMA_CHANNEL_ANY</code> to choose any available channel.
<b>rcmask</b>	Channel mask for DMA receive. Use <code>DMA_CHANNEL_ANY</code> to choose any available channel.

### RETURN VALUE

DMA error code or 0 for success

### LIBRARY

`RS232.LIB`

### SEE ALSO

`serXdmaOff`



---

---

## serXflowcontrolOff

---

---

```
void serXflowcontrolOff( void ); /* where X is A-F */
```

### DESCRIPTION

Turns off hardware flow control for serial port X. A call to `serXopen()` must be made before calling this function. This function is non-reentrant.

The functions `serEflowcontrolOff()` and `serFflowcontrolOff()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXflowcontrolOff(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### LIBRARY

RS232.LIB

---

---

## serXflowcontrolOn

---

---

```
void serXflowcontrolOn( void ); /* where X is A-F */
```

### DESCRIPTION

Turns on hardware flow control for channel X. This enables two digital lines that handle flow control, CTS (clear to send) and RTS (ready to send). CTS is an input that will be pulled active low by the other system when it is ready to receive data. The RTS signal is an output that the system uses to indicate that it is ready to receive data; it is driven low when data can be received. A call to `serXopen()` must be made before calling this function.

This function is non-reentrant.

The functions `serEflowcontrolOn()` and `serFflowcontrolOn()` may be used with the Rabbit 3000 and Rabbit 4000.

If pins for the flow control lines are not explicitly defined, defaults will be used and compiler warnings will be issued. The locations of the flow control lines are specified using a set of 5 macros.

<code>SERX_RTS_PORT</code>	Data register for the parallel port that the RTS line is on. e.g. PCDR
<code>SERA_RTS_SHADOW</code>	Shadow register for the RTS line's parallel port. e.g. PCDRShadow
<code>SERA_RTS_BIT</code>	The bit number for the RTS line
<code>SERA_CTS_PORT</code>	Data register for the parallel port that the CTS line is on
<code>SERA_CTS_BIT</code>	The bit number for the CTS line

### LIBRARY

`RS232.LIB`

---

---

## serXgetc

---

---

```
int serXgetc( void ); /* where X is A-F */
```

### DESCRIPTION

Get next available character from serial port X read buffer. This function is non-reentrant. The functions `serEgetc()` and `serFgetc()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXgetc(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

Success: the next character in the low byte, 0 in the high byte.  
Failure: -1, which indicates either an empty or a locked receive buffer.

### LIBRARY

`RS232.LIB`

### EXAMPLE

```
// echoes characters
main() {
    int c;
    serAopen(19200);
    while (1) {
        if ((c = serAgetc()) != -1) {
            serAputc(c);
        }
    }
    serAclose()
}
```

---

---

## serXgetError

---

---

```
int serXgetError( void ); /* where X is A-F */
```

### DESCRIPTION

Returns a byte of error flags, with bits set for any errors that occurred since the last time this function was called. Any bits set will be automatically cleared when this function is called, so a particular error will only be reported once. This function is non-reentrant.

The flags are checked with bitmasks to determine which errors occurred. Error bitmasks:

- SER\_PARITY\_ERROR
- SER\_OVERRUN\_ERROR

The functions serEgetError() and serFgetError() may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: serXgetError(int port), where “port” is one of the macros SER\_PORT\_A through SER\_PORT\_F.

### RETURN VALUE

The error flags byte.

### LIBRARY

RS232.LIB

---

---

## serXopen

---

---

```
int serXopen( long baud ); /* where X is A-F */
```

### DESCRIPTION

Opens serial port X. This function is non-reentrant.

The user must define the buffer sizes for each port being used with the buffer size macros XINBUFSIZE and XOUTBUFSIZE. The values must be a power of 2 minus 1, e.g.

```
#define XINBUFSIZE    63
#define XOUTBUFSIZE  127
```

Defining the buffer sizes to  $2^n - 1$  makes the circular buffer operations very efficient. If a value not equal to  $2^n - 1$  is defined, a default of 31 is used and a compiler warning is given.

The functions `serEopen()` and `serFopen()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** The alternate pins on parallel port D can be used for serial port B by defining `SERB_USEPORTD` at the beginning of a program. See the section on parallel port D in the Rabbit documentation for more detail on the alternate serial port pins.

For Rabbit 4000 users: To use DMA for transfers, call `serXdmaOn()` after this function.

### PARAMETERS

**baud**                      Bits per second (bps) of data transfer. Note that the baud rate must be greater than or equal to the peripheral clock frequency divided by 8192.

### RETURN VALUE

- 1: The Rabbit's bps setting is within 5% of the input baud.
- 0: The Rabbit's bps setting differs by more than 5% of the input baud.

### LIBRARY

RS232.LIB

### SEE ALSO

`serXgetc`, `serXpeek`, `serXputs`, `serXwrite`, `cof_serXgetc`,  
`cof_serXgets`, `cof_serXread`, `cof_serXputc`, `cof_serXputs`,  
`cof_serXwrite`, `serXclose`

---

---

## serXparity

---

---

```
void serXparity( int parity_mode ); /* where X is A-F */
```

### DESCRIPTION

Sets parity mode for channel X. A call to `serXopen()` must be made before calling this function.

Parity generation for 8-bit data can be unusually slow due to the current method for generating high 9th bits. Whenever a 9th high bit is needed, the UART is disabled for approximately 10 baud times to create a long stop bit that should be recognized by the receiver as a high 9th bit.

The long delay is imposed because we are using the serial port itself to handle timing for the delay. Creating a shorter delay would require use of some other timer resource.

This function is non-reentrant.

The functions `serEparity()` and `serFparity()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXparity(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

- parity\_mode** An integer indicating what parity mode to use. It is best to use one of the macros provided:
- `PARAM_NOPARITY` - Disables parity handling (default).
  - `PARAM_OPARITY` - Odd parity; parity bit set to “0” if odd number of 1’s in data bits.
  - `PARAM_EPARITY` - Even parity; parity bit set to “1” if even number of 1’s in data bits.
  - `PARAM_MPARITY` - Mark parity; parity bit always set to logical 1. (Rabbit 4000 only)
  - `PARAM_SPARITY` - Space parity; parity bit always set to logical 0. (Rabbit 4000 only)
  - `PARAM_2STOP` - 2 stop bits.

From a logical standpoint, the first three of these `PARAM_` macros cannot be combined, but even `PARAM_2STOP` must stand alone due to limitations in the UART hardware that will not allow parity bits and extra stop bits.

### LIBRARY

`RS232.LIB`

---

---

## serXpeek

---

---

```
int serXpeek( void ); /* where X is A-F */
```

### DESCRIPTION

Returns first character in input buffer X, without removing it from the buffer. This function is non-reentrant.

The functions `serEpeek()` and `serFpeek()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXpeek(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

An integer with first character in buffer in the low byte.  
-1 if the buffer is empty.

### LIBRARY

RS232.LIB

---

---

## serXputc

---

---

```
int serXputc( char c ); /* where X is A-F */
```

### DESCRIPTION

Writes a character to serial port X write buffer. This function is non-reentrant.

The functions `serEputc()` and `serFputc()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXputc(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

**c**                      Character to write to serial port X write buffer.

### RETURN VALUE

0 if buffer locked or full, 1 if character sent.

### LIBRARY

RS232.LIB

### EXAMPLE

```
main() { // echoes characters
    int c;
    serAopen(19200);
    while (1) {
        if ((c = serAgetc()) != -1) {
            serAputc(c);
        }
    }
    serAclose();
}
```



---

---

## serXputs

---

---

```
int serXputs( char * s ); /* where X is A-F */
```

### DESCRIPTION

Calls `serXwrite(s, strlen(s))`; does not write null terminator. This function is non-reentrant.

The functions `serEputs()` and `serFputs()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXputs(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

`s` Null terminated character string to write

### RETURN VALUE

The number of characters actually sent from serial port X.

### LIBRARY

`RS232.LIB`

### EXAMPLE

```
// writes a null-terminated string of characters, repeatedly
main() {
    const static char s[] = "Hello Rabbit";
    serAopen(19200);
    while (1) {
        serAputs(s);
    }
    serAclose();
}
```

---

---

## serXrdFlush

---

---

```
void serXrdFlush( void ); /* where X is A-F */
```

### DESCRIPTION

Flushes serial port X input buffer. This function is non-reentrant.

The functions `serErdfFlush()` and `serFrdfFlush()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXrdFlush(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### LIBRARY

`RS232.LIB`

---

---

## serXrdFree

---

---

```
int serXrdFree( void ); /* where X is A-F */
```

### DESCRIPTION

Calculates the number of characters of unused data space. This function is non-reentrant.

The functions `serErdfFree()` and `serFrdfFree()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXrdFree(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

The number of chars it would take to fill input buffer X.

### LIBRARY

`RS232.LIB`

---

---

## serXrdUsed

---

---

```
int serXrdUsed( void ); /* where X is A-F */
```

### DESCRIPTION

Calculates the number of characters ready to read from the serial port receive buffer. This function is non-reentrant.

The functions `serErdUsed()` and `serFrdUsed()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXrdUsed(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

The number of characters currently in serial port X receive buffer.

### LIBRARY

`RS232.LIB`

---

---

## serXread

---

---

```
int serXread( void * data, int length, unsigned long tmout );
/* where X is A-F */
```

### DESCRIPTION

Reads `length` bytes from serial port `X` or until `tmout` milliseconds transpires between bytes. The countdown of `tmout` does not begin until a byte has been received. A timeout occurs immediately if there are no characters to read. This function is non-reentrant.

The functions `serEread()` and `serFread()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXread(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

<b>data</b>	Data structure to read from serial port X
<b>length</b>	Number of bytes to read
<b>tmout</b>	Maximum wait in milliseconds for any byte from previous one

### RETURN VALUE

The number of bytes read from serial port X.

### LIBRARY

RS232.LIB

### EXAMPLE

```
// echoes a blocks of characters
main() {
    int n;
    char s[16];
    serAopen(19200);
    while (1) {
        if ((n = serAread(s, 15, 20)) > 0) {
            serAwrite(s, n);
        }
    }
    serAclose();
}
```

---

---

## `serXwrFlush`

---

---

```
void serXwrFlush( void ); /* where X is A-F */
```

### DESCRIPTION

Flushes serial port X transmit buffer, meaning that the buffer contents will not be sent. This function is non-reentrant.

The functions `serEwrFlush()` and `serFwrFlush()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXwrFlush(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### LIBRARY

`RS232.LIB`

---

---

## `serXwrFree`

---

---

```
int serXwrFree( void ); /* where X is A-F */
```

### DESCRIPTION

Calculates the free space in the serial port transmit buffer. This function is non-reentrant.

The functions `serEwrFree()` and `serFwrFree()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXwrFree(port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

The number of characters the serial port transmit buffer can accept before becoming full.

### LIBRARY

`RS232.LIB`

---

---

## serXwrite

---

---

```
int serXwrite( void * data, int length ); /* X is A-F */
```

### DESCRIPTION

Transmits `length` bytes to serial port X. This function is non-reentrant.

The functions `serEwrite()` and `serFwrite()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXwrite(int port, ...)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### PARAMETERS

<code>data</code>	Data structure to write to serial port X
<code>length</code>	Number of bytes to write

### RETURN VALUE

The number of bytes successfully written to the serial port.

### LIBRARY

`RS232.LIB`

### EXAMPLE

```
// writes a block of characters, repeatedly
main() {
    const char s[] = "Hello Rabbit";
    serAopen(19200);
    while (1) {
        serAwrite(s, strlen(s));
    }
    serAclose();
}
```

---

---

## serXwrUsed

---

---

```
int serXwrUsed( void ); /* where X is A-F */
```

### DESCRIPTION

Returns the number of characters in the output buffer. This function is non-reentrant.

The functions `serErdUsed()` and `serFrdUsed()` may be used with the Rabbit 3000 and Rabbit 4000.

**Note:** Alternatively you can use another form of this function that has been generalized for all serial ports. Instead of substituting for “X” in the function name, the prototype of the generalized function is: `serXwrUsed(int port)`, where “port” is one of the macros `SER_PORT_A` through `SER_PORT_F`.

### RETURN VALUE

The number of characters currently in the output buffer.

### LIBRARY

RS232.LIB

---

---

## set

---

---

```
void set( void * address, unsigned int bit );
```

### DESCRIPTION

Dynamic C may expand this call inline. Sets specified bit at memory address to 1. bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address |= 1L << bit
```

### PARAMETERS

<b>address</b>	Address of byte containing bits 7-0
<b>bit</b>	Bit location where 0 represents the least significant bit

### LIBRARY

UTIL.LIB

### SEE ALSO

SET

---

---

## SET

---

---

```
void SET( void * address, unsigned int bit );
```

### DESCRIPTION

Dynamic C may expand this call inline. Sets specified bit at memory address to 1. bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address |= 1L << bit
```

### PARAMETERS

<b>address</b>	Address of byte containing bits 7-0.
<b>bit</b>	Bit location where 0 represents the least significant bit.

### LIBRARY

UTIL.LIB

### SEE ALSO

set



---

---

## set32kHzDivider

---

---

```
void set32kHzDivider( int setting );
```

### DESCRIPTION

Sets the expanded 32kHz oscillator divider for the Rabbit 3000 processor. This function does not enable running the 32kHz oscillator instead of the main clock. This function will affect the actual rate used by the processor when the 32kHz oscillator has been enabled to run by a call to `use32kHzOsc()`.

This function is not task reentrant.

### PARAMETER

**setting**            32kHz divider setting. The following are valid:

- OSC32DIV\_1 - don't divide 32kHz oscillator
- OSC32DIV\_2 - divide 32kHz oscillator by two
- OSC32DIV\_4 - divide 32kHz oscillator by four
- OSC32DIV\_8 - divide 32kHz oscillator by eight
- OSC32DIV\_16 - divide 32kHz oscillator by sixteen

### LIBRARY

`SYS.LIB`

### SEE ALSO

`useClockDivider`, `useClockDivider3000`, `useMainOsc`, `use32kHzOsc`

---

---

## setClockModulation

---

---

```
void setClockModulation( int setting );
```

### DESCRIPTION

Changes the setting of the Rabbit 3000 CPU clock modulation. Calling this function will force a 500 clock delay before the setting is changed to ensure that the previous modulation setting has cleared before the next one is set. See the *Rabbit 3000 Microprocessor User's Manual* for more details about clock modulation for EMI reduction.

### PARAMETER

**setting**            Clock modulation setting. Allowed values are:

- 0 = no modulation
- 1 = weak modulation
- 2 = strong modulation

### LIBRARY

SYS.LIB

---

---

## set\_cpu\_power\_mode

---

---

```
int set_cpu_power_mode( int mode, char clkDoubler, char
    shortChipSelect );
```

### DESCRIPTION

Sets operating power of the controller. Suspend serial communication and other data transmission activity prior to calling this function, which sets higher priority interrupt while switching clock frequencies.

This function is non-reentrant.

### PARAMETERS

**mode** Mode operation. Use the following table values below. (The higher the value the lower the power consumption of controller.)

Mode	Description	Comments
1	Cclk=Pclk=MainOsc	Debug capable
2	Cclk=Pclk=MainOsc/2	Debug capable (19200 baud)
3	Cclk=Pclk=MainOsc/4	Debug capable (9600 baud)
4	Cclk=Pclk=MainOsc/6	
5	Cclk=Pclk=MainOsc/8	
6	Cclk=Pclk= 32.768KHz	Periodic Interrupt disabled, so call hitwd()
7	Cclk=Pclk=32KHz/2=16.384KHz	Periodic Interrupt disabled, so call hitwd()
8	Cclk=Pclk=32KHz/4 =8.192KHz	Periodic Interrupt disabled, so call hitwd()
9	Cclk=Pclk=32KHz/8=4.096KHz	Periodic Interrupt disabled, so call hitwd()
10	Cclk=Pclk=32kHz/16 =2.048KHz	Periodic Interrupt disabled, so call hitwd()

---

---

## set\_cpu\_power\_mode (cont'd)

---

---

**clkDoubler**      Clock doubler setting: CLKDOUBLER\_ON or CLKDOUBLER\_OFF.  
CPU will operate at half selected speed when turned off. This parameter only affects main oscillator modes, not 32 kHz oscillator modes. Turning Clock doubler off reduces power consumption.

**shortChipSelect**      Short Chip Select setting. Use SHORTCS\_OFF, or SHORTCS\_ON.

**Note:** When short chip select is on, make sure that interrupts are disabled during I/O operations. Turning Short Chip Select on may reduce power consumption. See the Rabbit processor manual for more information regarding chip selects and low power operation.

### RETURN VALUE

0: valid parameter  
-1: invalid parameter

### LIBRARY

low\_power.lib

---

---

## setjmp

---

---

```
int setjmp( jmp_buf env );
```

### DESCRIPTION

Store the PC (program counter), SP (stack pointer) and other information about the current state into `env`. The saved information can be restored by executing `longjmp()`.

**Note:** you cannot use `setjmp()` to move out of slice statements, costatements, or cofunctions.

Typical usage:

```
switch (setjmp(e)) {
    case 0:          // first time
        f();         // try to execute f(), may call longjmp()
        break;      // if we get here, f() was successful
    case 1:          // to get here, f() called longjmp()
        /* do exception handling */
        break;
    case 2:          // similar to above, but different exception code
        ...
}
f() {
    g()
    ...
}
g() {
    ...
    longjmp(e, 2); // exception code 2, jump to setjmp() statement,
                  // setjmp() returns 2, so execute
                  // case 2 in the switch statement
}
```

### PARAMETERS

**env** Information about the current state

### RETURN VALUE

Returns zero if it is executed. After `longjmp()` is executed, the program counter, stack pointer and etc. are restored to the state when `setjmp()` was executed the first time. However, this time `setjmp()` returns whatever value is specified by the `longjmp()` statement.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`longjmp`

---

---

## SetSerialTATxRValues

---

---

```
long SetSerialTATxRValues( long bps, char *divisor, int tatXr );
```

### DESCRIPTION

Sets up the possibly shared serial timer (TATxR) resources required to achieve, as closely as possible, the requested serial bps rate. The algorithm attempts to find, when necessary and if possible, the lowest value for the TAT1R that will precisely produce the requested serial bps rate. For this reason, an application that requires the TAT1R to be shared should generally first set up its usage with (1) the most critical timer A1 cascade rate, or (2) the lowest timer A1 cascade rate. That is, consider setting up the most critical stage (PWM, servo, triac, ultra-precise serial rate, etc.) first, else set up the slowest usage (often, the lowest serial rate) first.

Note that this function provides no TATxR resource sharing protection for an application that uses any of the individual TATxR resources either directly or indirectly. For example, this function affords no protection to an application that sets a direct usage TAT7R timer interrupt and also opens serial port D such that TAT7R is used to set the serial data rate.

A run time error occurs if parameter(s) are invalid. Also, this function is not reentrant.

### PARAMETERS

<b>bps</b>	The requested serial bits per second (BPS, baud) rate.
<b>divisor</b>	An optional pointer to the caller's serial timer divisor variable. If the caller is not interested in the actual serial timer (TATxR) divisor value that is set by this function, then NULL may be passed.
<b>tatXr</b>	The TATxR for the serial timer whose value(s) are to be set. Use exactly one of the following macros: <ul style="list-style-type: none"><li>• TAT4R for serial port A</li><li>• TAT5R for serial port B</li><li>• TAT6R for serial port C</li><li>• TAT7R for serial port D</li><li>• TAT2R for serial port E</li><li>• TAT3R for serial port F</li></ul>

### RETURN VALUE

The actual serial rate BPS (baud) setting that was achieved.

### LIBRARY

sys.lib

### SEE ALSO

TAT1R\_SetValue

---

---

## SetVectExtern2000

---

---

```
unsigned SetVectExtern2000( int priority, void * isr );
```

### DESCRIPTION

Sets up the external interrupt table vectors for external interrupts 0 and 1. This function should be used for Rabbit 2000 processors revision IQ2 due to a bug in the chip's interrupt handling. (See Technical Note 301, "Rabbit 2000 Microprocessor Interrupt Issue," on the [Rabbit Semiconductor website](#) for more information.)

Once this function is called, both external interrupts 0 and 1 should be enabled with priority levels set higher than any currently running interrupts. (All system interrupts in the BIOS run at interrupt priority 1.) The interrupt priority is set via the control register IOCR for external interrupt 0 and IICR for external interrupt 1.

The actual priority used by the interrupt service routine (ISR) is passed to this function.

### PARAMETERS

<b>priority</b>	Priority the ISR should run at. Valid values are 1, 2 or 3.
<b>isr</b>	ISR handler address. Must be a root address.

### RETURN VALUE

Address of vector table entry, or zero if `priority` is not valid.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`GetVectExtern2000`, `SetVectIntern`, `GetVectIntern`

---

---

## SetVectExtern3000

---

---

```
unsigned SetVectExtern3000( int interruptNum, void * isr );
```

### DESCRIPTION

Function to set one of the external interrupt jump table entries for the Rabbit 3000 and some versions of the Rabbit 2000. All Rabbit interrupts use jump vectors. See `SetVectIntern()` for more information.

### PARAMETERS

**interruptNum** External interrupt number. 0 and 1 are the only valid values.

**isr** ISR handler address. Must be a root address.

### RETURN VALUE

Jump address in vector table.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`GetVectExtern3000`, `SetVectIntern`, `GetVectIntern`



---

---

## SetVectExtern4000

---

---

```
unsigned SetVectExtern4000( int interruptNum, void * isr );
```

### DESCRIPTION

Function to set one of the external interrupt jump table entries for the Rabbit 4000, Rabbit 3000 and some versions of the Rabbit 2000. All Rabbit interrupts use jump vectors. See `SetVectIntern()` for more information.

### PARAMETERS

**interruptNum** External interrupt number. 0 and 1 are the only valid values.

**isr** ISR handler address. Must be a root address.

### RETURN VALUE

Jump address in vector table.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`GetVectExtern3000`, `SetVectIntern`, `GetVectIntern`

---

---

## SetVectIntern

---

---

```
unsigned SetVectIntern( int vectNum, void * isr );
```

### DESCRIPTION

Sets an internal interrupt table entry. All Rabbit interrupts use jump vectors. This function writes a `jp` instruction (0xC3) followed by the 16 bit ISR address to the appropriate location in the vector table. The location in RAM of the vector table is determined and set by the BIOS automatically at startup. The start of the table is always on a 0x100 boundary.

It is perfectly permissible to have ISRs in `xmem` and do long jumps to them from the vector table. It is even possible to place the entire body of the ISR in the vector table if it is 16 bytes long or less, but this function only sets up jumps to 16 bit addresses.

The following table shows the `vectNum` value for each peripheral or RST. The offset into the vector table is also shown. The following vectors are valid for all Rabbit processors.

Peripheral or RST	vectNum	Vector Table Offset
Periodic interrupt	0x00	0x00
RST 10 instruction	0x02	0x20
RST 38 instruction	0x07	0x70
Slave Port	0x08	0x80
Timer A	0x0A	0xA0
Timer B	0x0B	0xB0
Serial Port A	0x0C	0xC0
Serial Port B	0x0D	0xD0
Serial Port C	0x0E	0xE0
Serial Port D	0x0F	0xF0

The following vectors are valid starting with the Rabbit 3000.

Peripheral or RST	vectNum	Vector Table Offset
Input Capture	0x1A	0x01A0
Quadrature Encoder	0x19	0x0190
Serial port E	0x1C	0x01C0
Serial port F	0x1D	0x01D0

---

---

## SetVectIntern (cont'd)

---

---

The following vectors are valid starting with the Rabbit 3000 Revision 1.

Peripheral or RST	vectNum	Vector Table Offset
Pulse Width Modulator	0x17	0x0170
Secondary Watchdog	0x01	0x10

The following vectors are valid starting with the Rabbit 4000.

Peripheral or RST	vectNum	Vector Table Offset
Timer C	0x1F	0x01F0
Network Port A	0x1E	0x01E0

The following three RSTs are included for completeness, but should not be set by the user as they are used by Dynamic C.

Peripheral or RST	vectNum	Vector Table Offset
RST 18 instruction	0x03	0x30
RST 20 instruction	0x04	0x40
RST 28 instruction	0x05	0x50

### PARAMETERS

**vectNum**            Interrupt number. See the above table for valid values.  
**isr**                 ISR handler address. Must be a root address.

### RETURN VALUE

Address of vector table entry, or zero if `vectNum` is not valid.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`GetVectExtern2000`, `SetVectExtern2000`, `GetVectIntern`

---

---

## **sf\_getPageCount**

---

---

```
long sf_getPageCount( sf_device * dev );
```

### **DESCRIPTION**

Return number of pages in a flash device.

### **PARAMETER**

**dev**                    Pointer to `sf_device` struct for initialized flash device.

### **RETURN VALUE**

Number of pages.

### **LIBRARY**

SFLASH.LIB

---

---

## **sf\_getPageSize**

---

---

```
unsigned int sf_getPageSize( sf_device * dev );
```

### **DESCRIPTION**

Return size (in bytes) of a page on the current flash device.

### **PARAMETER**

**dev**                    Pointer to `sf_device` struct for initialized flash device.

### **RETURN VALUE**

Bytes in a page.

### **LIBRARY**

SFLASH.LIB

---

---

## `sf_init`

---

---

```
int sf_init( void );
```

### DESCRIPTION

Initializes serial flash chip. This function must be called before the serial flash can be used. Currently supported devices are:

- AT45DB041
- AT45DB081
- AT45DB642
- AR45DB1282

**Note:** This function blocks and only works on boards with one serial flash device.

### RETURN VALUE

- 0 for success
- 1 if no flash chip detected
- 2 if error communicating with flash chip
- 3 if unknown flash chip type

### LIBRARY

`SFLASH.LIB`

---

---

## `sf_initDevice`

---

---

```
int sf_initDevice( sf_device * dev, int cs_port, char * cs_shadow,
                  int cs_pin );
```

### DESCRIPTION

Replaces `sf_init()`.

The function `sfspi_init()` must be called before any calls to this function. Initializes serial flash chip. This function must be called before the serial flash can be used. Currently supported devices are:

- AT45DB041
- AT45DB081
- AT45DB642
- AR45DB1282

### PARAMETERS

<code>dev</code>	Pointer to an empty <code>sf_device</code> struct that will be filled in on return. The struct will then act as a handle for the device.
<code>cs_port</code>	I/O port for the active low chip select pin for the device.
<code>cs_shadow</code>	Pointer to the shadow variable for <code>cs_port</code> .
<code>cs_pin</code>	I/O port pin number for the chip select signal.

### RETURN VALUE

- 0 for success
- 1 if no flash chip detected
- 2 if error communicating with flash chip
- 3 if unknown flash chip type

### LIBRARY

`SFLASH.LIB`

---

---

## `sf_isWriting`

---

---

```
int sf_isWriting( sf_device * dev );
```

### DESCRIPTION

Returns 1 if the flash device is busy writing to a page.

### PARAMETER

**dev**                      Pointer to `sf_device` struct for initialized flash device

### RETURN VALUE

1 busy  
0 ready, not currently writing

### LIBRARY

`SFLASH.LIB`

---

---

## `sf_pageToRAM`

---

---

```
int sf_pageToRAM( long page );
```

### DESCRIPTION

Command the serial flash to copy the contents of one of its flash pages into its RAM buffer.

**Note:** This function blocks and only works on boards with one serial flash device.

### PARAMETER

**page**                      The page to copy.

### RETURN VALUE

0 for success  
-1 for error

### LIBRARY

`SFLASH.LIB`

---

---

## sf\_RAMToPage

---

---

```
int sf_RAMToPage( long page );
```

### DESCRIPTION

Command the serial flash to write its RAM buffer contents to one of the flash memory pages.

**Note:** This function blocks and only works on boards with one serial flash device.

### PARAMETER

**page**                    The page to which the RAM buffer contents will be written t

### RETURN VALUE

0 for success  
-1 for error

### LIBRARY

SFLASH.LIB



---

---

## `sf_readDeviceRAM`

---

---

```
int sf_readDeviceRAM( sf_device * dev, long buffer, int offset,
    int len, int flags );
```

### DESCRIPTION

Read data from the RAM buffer on the serial flash chip into an xmem buffer.

### PARAMETERS

<code>dev</code>	Pointer to <code>sf_device</code> struct for initialized flash device.
<code>buffer</code>	Address of an xmem buffer.
<code>offset</code>	The address in the serial flash RAM to start reading from.
<code>len</code>	The number of bytes to read.
<code>flags</code>	Can be one of the following:  <code>SF_BITSREVERSED</code> - Reads the data in bit reversed order from the flash chip. This improves speed, but the data must have been also written in reversed order (see <code>sf_XWriteRAM</code> )  <code>SF_RAMBANK1</code> (default) - Reads from the first RAM bank on the flash device  <code>SF_RAMBANK2</code> - Reads from the alternate RAM bank on the flash device

### RETURN VALUE

0: Success  
-1: Error

### LIBRARY

`SFLASH.LIB`

---

---

## **sf\_readPage**

---

---

```
int sf_readPage( sf_device * dev, int bank, long page );
```

### **DESCRIPTION**

Replaces `sf_pageToRAM()`.

Command the serial flash to copy from one of its flash pages to one of its RAM buffers.

### **PARAMETERS**

<b>dev</b>	Pointer to <code>sf_device</code> struct for initialized flash device.
<b>bank</b>	Which RAM bank to write the data to. For Atmel 45DBxxx devices, this can be 1 or 2.
<b>page</b>	The page to read from.

### **RETURN VALUE**

0: Success  
-1: Error

### **LIBRARY**

SFLASH.LIB

---

---

## **sf\_readRAM**

---

---

```
int sf_readRAM( char * buffer, int offset, int len );
```

### **DESCRIPTION**

Read data from the RAM buffer on the serial flash chip.

**Note:** This function blocks and only works on boards with one serial flash device.

### **PARAMETER**

<b>buffer</b>	Pointer to character buffer to copy data into.
<b>offset</b>	Address in the serial flash RAM to start reading from
<b>len</b>	Number of bytes to read

### **RETURN VALUE**

0: Success  
-1: Error

### **LIBRARY**

SFLASH.LIB

---

---

## `sf_writeDeviceRAM`

---

---

```
int sf_writeDeviceRAM( sf_device * dev, long buffer, int offset,
    int len, int flags );
```

### DESCRIPTION

Write data to the RAM buffer on the serial flash chip from a buffer in xmem.

### PARAMETER

<b>dev</b>	Pointer to <code>sf_device</code> struct for initialized flash device.
<b>buffer</b>	Pointer to xmem data to write into the flash chip RAM.
<b>offset</b>	The address in the serial flash RAM to start writing at.
<b>len</b>	The number of bytes to write.
<b>flags</b>	Can be one of the following: <ul style="list-style-type: none"><li>• <code>SF_BITSREVERSED</code> - Allows the data to be written to the flash in reverse bit order. This improves speed, and works fine as long as the data is read back out with this same flag (see <code>sf_XReadRAM</code>)</li><li>• <code>SF_RAMBANK1</code> (default) - Writes to the first RAM bank on the flash device</li><li>• <code>SF_RAMBANK2</code> - Writes to the alternate RAM bank on the flash device</li></ul>

### RETURN VALUE

0: Success  
-1: Error

### LIBRARY

`SFLASH.LIB`

---

---

## sf\_writePage

---

---

```
int sf_writePage( sf_device * dev, int bank, long page );
```

### DESCRIPTION

Replaces `sf_RAMToPage()`.

Command the serial flash to write its RAM buffer contents to one of its flash memory pages. Check for completion of the write operation using `sf_isWriting()`.

### PARAMETERS

<b>dev</b>	Pointer to <code>sf_device</code> struct for initialized flash device.
<b>bank</b>	Which RAM bank to write the data from. For Atmel 45DBxxx devices, this can be 1 or 2
<b>page</b>	The page to write the RAM buffer to

### RETURN VALUE

0: Success  
-1: Error

### LIBRARY

SFLASH.LIB

---

---

## sf\_writeRAM

---

---

```
int sf_writeRAM( char * buffer, int offset, int len );
```

### DESCRIPTION

Write data to the RAM buffer on the serial flash chip.

**Note:** This function blocks and only works on boards with one serial flash device.

### PARAMETER

<b>buffer</b>	Pointer to data that will be written the flash chip RAM.
<b>offset</b>	Address in the serial flash RAM to start writing at.
<b>len</b>	Number of bytes to write.

### RETURN VALUE

0 for success  
-1 for error

### LIBRARY

SFLASH.LIB

---

---

## sfspi\_init

---

---

```
int sfspi_init( void );
```

### DESCRIPTION

Initialize SPI driver for use with serial flash. This must be called before any calls to `sf_initDevice()`.

### RETURN VALUE

0 for success  
-1 for error

### LIBRARY

SFLASH.LIB

---

---

## sin

---

---

```
float sin ( float x );
```

### DESCRIPTION

Computes the sine of  $x$ .

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

**x**                      Angle in radians.

### RETURN VALUE

Sine of  $x$ .

### LIBRARY

MATH.LIB

### SEE ALSO

`sinh`, `asin`, `cos`, `tan`

---

---

## sinh

---

---

```
float sinh( float x );
```

### DESCRIPTION

Computes the hyperbolic sine of  $x$ . This functions takes a unitless number as a parameter and returns a unitless number.

### PARAMETERS

**x**                      Value to compute.

### RETURN VALUE

The hyperbolic sine of  $x$ .

If  $x > 89.8$  (approx.), the function returns INF and signals a range error. If  $x < -89.8$  (approx.), the function returns -INF and signals a range error.

### LIBRARY

MATH.LIB

### SEE ALSO

`sin`, `asin`, `cosh`, `tanh`

---

---

## snprintf

---

---

```
int snprintf( char * buffer, int len, char * format, ... );
```

### DESCRIPTION

This function takes a string (pointed to by `format`), arguments of the format, and outputs the formatted string to the buffer pointed to by `buffer`. `snprintf()` will only output up to `len` characters. The user should make sure that:

- there are enough arguments after `format` to fill in the format parameters in the format string
- the types of arguments after `format` match the format fields in `format`

For example,

```
snprintf(buffer, BUF_LEN, "%s=%x", "variable x", 256);
```

puts the string “variable x=100” into `buffer`.

A complete list of valid conversion specifiers (`%d`, `%s`, etc.) can be found in the description for `printf()` under Dynamic C Conversion Specifiers.

The macro `STDIO_DISABLE_FLOATS` can be defined if it is not necessary to format floating point numbers. If this macro is defined, `%e`, `%f` and `%g` will not be recognized. This can save thousands of bytes of code space.

This function can be called by processes of different priorities.

### PARAMETERS

<b>buffer</b>	Location of formatted string.
<b>len</b>	The maximum length of the formatted string.
<b>format</b>	String to be formatted.
<b>...</b>	Format arguments.

### RETURN VALUE

The number of characters written. If the output is truncated due to the `len` parameter, then this function returns the number of characters that would have been written had there been enough space.

### LIBRARY

`STDIO.LIB`

### SEE ALSO

`printf`, `sprintf`



---

---

## SPIinit

---

---

```
void SPIinit( void );
```

### DESCRIPTION

Initialize the SPI port parameters for a serial interface only. This function does nothing for a parallel interface. A description of the values that the user may define before the `#use SPI.LIB` statement is found at the top of the library `Lib\Spi\Spi.lib`.

### LIBRARY

`SPI.LIB`

### SEE ALSO

`SPIRead`, `SPIWrite`, `SPIWrRd`

---

---

## SPIRead

---

---

```
void SPIRead( void * DestAddr, int ByteCount );
```

### DESCRIPTION

Reads a block of bytes from the SPI port. The variable `SPIxor` needs to be set to either 0x00 or 0xFF depending on whether or not the received signal needs to be inverted. Most applications will not need inversion. `SPIinit()` sets the value of `SPIxor` to 0x00.

If `SPI_SLAVE_RDY_PORT` is defined for a slave device the driver will turn on the bit immediately upon activating the receiver. It will then wait for a byte to become available then turn off the bit. The byte will not be available until the master supplies the 8 clock pulses.

If `SPI_SLAVE_RDY_PORT` is defined for a master device the driver will wait for the bit to become true before activating the receiver and then wait for it to become false after receiving the byte.

Note for Master: the receiving device Chip Select must already be active

### PARAMETERS

<b>DestAddr</b>	Address to store the data
<b>ByteCount</b>	Number of bytes to read

### RETURN VALUE

Master: none.

Slave: 0 = no CS signal, no received bytes.

1 = CS, bytes received.

### LIBRARY

`SPI.LIB`

### SEE ALSO

`SPIinit`, `SPIwrite`, `SPIWrRd`

---

---

## SPIWrite

---

---

```
int SPIWrite( void * SrcAddr, int ByteCount );
```

### DESCRIPTION

Write a block of bytes to the SPI port.

If `SPI_SLAVE_RDY_PORT` is defined for a slave device the driver will turn on the bit immediately after loading the transmit register. It will then wait for the buffer to become available then turn off the bit. The buffer will not become available until the master supplies the first clock.

If `SPI_SLAVE_RDY_PORT` is defined for a master device the driver will wait for the bit to become true before transmitting the byte and then wait for it to become false after transmitting the byte.

Note for Master: the receiving device Chip Select must already be active.

### PARAMETERS

<b>SrcAddr</b>	Address of data to write.
<b>ByteCount</b>	Number of bytes to write.

### RETURN VALUE

Master: none.

Slave: 0 = no CS signal, no transmitted bytes.

1 = CS, bytes transmitted.

### LIBRARY

`SPI.LIB`

### SEE ALSO

`SPIinit`, `SPIRead`, `SPIWrRd`

---

---

## SPIWrRd

---

---

```
void SPIWrRd( void * SrcAddr, void * DstAddr, int ByteCount );
```

### DESCRIPTION

Read and Write a block of bytes from/to the SPI port.

Note for Master: the receiving device Chip Select must already be active.

### PARAMETERS

<b>SrcAddr</b>	Address of data to write.
<b>DstAddr</b>	Address to put received data.
<b>ByteCount</b>	Number of bytes to read/write. The maximum value is 255 bytes. This limit is not checked! The receive buffer <b>MUST</b> be at least as large as the number of bytes!

### RETURN VALUE

Master: none.

Slave: 0 = no CS signal, no received/transmitted bytes.

1 = CS, bytes received/transmitted.

### LIBRARY

SPI.LIB

### SEE ALSO

SPIinit, SPIRead, SPIWrite

---

---

## sprintf

---

---

```
int sprintf( char * buffer, char * format, ... );
```

### DESCRIPTION

This function takes a string (pointed to by `format`), arguments of the format, and outputs the formatted string to `buffer` (pointed to by `buffer`). The user should make sure that:

- there are enough arguments after `format` to fill in the format parameters in the format string
- the types of arguments after `format` match the format fields in `format`
- the buffer is large enough to hold the longest possible formatted string

The following is a short list of valid conversion specifiers in the format string. For a complete list of conversion specifiers, refer to the function description for `printf()`.

**%d** decimal integer (expects type `int`)  
**%u** decimal unsigned integer (expects type `unsigned int`)  
**%x** hexadecimal integer (expects type `signed int` or `unsigned int`)  
**%s** a string (not interpreted, expects type `(char *)`)  
**%f** a float (expects type `float`)

For example,

```
printf(buffer, "%s = %x", "variable x", 256);
```

puts the string “variable x = 100” into `buffer`.

The macro `STDIO_DISABLE_FLOATS` can be defined if it is not necessary to format floating point numbers. If this macro is defined, `%e`, `%f` and `%g` will not be recognized. This can save thousands of bytes of code space.

This function can be called by processes of different priorities.

### PARAMETERS

<b>buffer</b>	Result string of the formatted string.
<b>format</b>	String to be formatted.
<b>...</b>	Format arguments.

### RETURN VALUE

Number of characters written.

### LIBRARY

`STDIO.LIB`

### SEE ALSO

`printf`

---

---

## sqrt

---

---

```
float sqrt( float x );
```

### DESCRIPTION

Calculate the square root of  $x$ .

### PARAMETERS

**x**                    Value to compute.

### RETURN VALUE

The square root of  $x$ .

### LIBRARY

MATH.LIB

### SEE ALSO

exp, pow, pow10

---

---

## srand

---

---

```
void srand( unsigned long seed );
```

### DESCRIPTION

Sets the seed value for the `rand()` function.

### PARAMETER

**seed**                    This must be an odd number.

### LIBRARY

MATH.LIB

### SEE ALSO

rand, randb, randg

---

---

## strcat

---

---

**NEAR SYNTAX:** `char * _n_strcat( char * dst, char * src );`  
**FAR SYNTAX:** `char far * _f_strcat( char far * dst, char far * src );`

**Note:** By default, `strcat()` is defined to `_n_strcat()`.

### DESCRIPTION

Concatenate string `src` to the end of `dst`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>dst</b>	Pointer to location to destination string.
<b>src</b>	Pointer to location to source string.

### RETURN VALUE

Pointer to destination string.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strncat`

---

---

## strchr

---

---

**NEAR SYNTAX:** `char * _n_strchr( char * src, char ch );`

**FAR SYNTAX:** `char far * _f_strchr( char far * src, char ch );`

**Note:** By default, `strchr()` is defined to `_n_strchr()`.

### DESCRIPTION

Scans a string for the first occurrence of a given character.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>src</b>	String to be scanned.
<b>ch</b>	Character to search

### RETURN VALUE

Pointer to the first occurrence of `ch` in `src`.  
Null if `ch` is not found.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strrchr`, `strtok`



---

---

## strcmp

---

---

**NEAR SYNTAX:** `int _n_strcmp( char * str1, char * str2 );`  
**FAR SYNTAX:** `int _f_strcmp( char far * str1, char far * str2 );`

**Note:** By default, `strcmp()` is defined to `_n_strcmp()`.

### DESCRIPTION

Performs unsigned character by character comparison of two null terminated strings.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**str1**            Pointer to string 1.

**str2**            Pointer to string 2.

### RETURN VALUE

<0: `str1` is less than `str2` because character in `str1` is less than corresponding character in `str2`, or `str1` is shorter than but otherwise identical to `str2`.

=0: `str1` is identical to `str2`

>0: `str1` is greater than `str2` because character in `str1` is greater than corresponding character in `str2`, or `str2` is shorter than but otherwise identical to `str1`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strncmp`, `strcmpi`, `strncmpi`

---

---

## strcmpi

---

---

**NEAR SYNTAX:** `int * _n_strcmpi( char * str1, char * str2 );`  
**FAR SYNTAX:** `int _f_strcmpi( char far * str1, char far * str2 );`

**Note:** By default, `strcmpi()` is defined to `_n_strcmpi()`.

### DESCRIPTION

Performs case-insensitive unsigned character by character comparison of two null terminated strings.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>str1</b>	Pointer to string 1.
<b>str2</b>	Pointer to string 2.

### RETURN VALUE

<0: `str1` is less than `str2` because character in `str1` is less than corresponding character in `str2`, or `str1` is shorter than but otherwise identical to `str2`.

=0: `str1` is identical to `str2`

>0: `str1` is greater than `str2` because character in `str1` is greater than corresponding character in `str2`, or `str2` is shorter than but otherwise identical to `str1`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strncmpi`, `strncmp`, `strcmp`

---

---

## strcpy

---

---

**NEAR SYNTAX:** `char * _n_strcpy( char * dst, char * src );`

**FAR SYNTAX:** `char far * _f_strcpy( char far * dst, char far * src );`

**Note:** By default, `strcpy()` is defined to `_n_strcpy()`.

### DESCRIPTION

Copies one string into another string, including the null terminator.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**dst**                      Pointer to location to receive string.

**src**                        Pointer to location to supply string.

### RETURN VALUE

Pointer to destination string.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strncpy`

---

---

## strcspn

---

---

**NEAR SYNTAX:** `unsigned int _n_strcspn( char * s1, char * s2 );`  
**FAR SYNTAX:** `size_t _f_strcspn( char far * s1, char far * s2 );`

**Note:** By default, `strcspn()` is defined to `_n_strcspn()`.

### DESCRIPTION

Scans a string for the occurrence of any of the characters in another string.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>s1</b>	String to be scanned.
<b>s2</b>	Character occurrence string.

### RETURN VALUE

Returns the position (less one) of the first occurrence of a character in `s1` that matches any character in `s2`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strchr`, `strrchr`, `strtok`

---

---

## strlen

---

---

**NEAR SYNTAX:** `int _n_strlen( char * s );`

**FAR SYNTAX:** `int _f_strlen( char far * s );`

**Note:** By default, `strlen()` is defined to `_n_strlen()`.

### DESCRIPTION

Calculate the length of a string.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**s**                      Character string.

### RETURN VALUE

Number of bytes in a string.

### LIBRARY

`STRING.LIB`

---

---

## strncat

---

---

**NEAR SYNTAX:** `char *_n_strncat( char *dst, char *src, unsigned int n );`  
**FAR SYNTAX:** `char far * _f_strncat( char far * dst, char far * src, size_t n );`

**Note:** By default, `strncat()` is defined to `_n_strncat()`.

### DESCRIPTION

Appends one string to another up to and including the null terminator or until `n` characters are transferred, followed by a null terminator.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>dst</b>	Pointer to location to receive string.
<b>src</b>	Pointer to location to supply string.
<b>n</b>	Maximum number of bytes to copy. If equal to zero, this function has no effect.

### RETURN VALUE

Pointer to destination string.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strcat`

---

---

## strncmp

---

---

```
NEAR SYNTAX: int _n_strncmp( char * str1, char * str2, n );
FAR SYNTAX: int _f_strncmp( char far * str1, char far * str2, unsigned
n );
```

**Note:** By default, `strncmp()` is defined to `_n_strncmp()`.

### DESCRIPTION

Performs unsigned character by character comparison of two strings of length `n`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<code>str1</code>	Pointer to string 1.
<code>str2</code>	Pointer to string 2.
<code>n</code>	Maximum number of bytes to compare. If zero, both strings are considered equal.

### RETURN VALUE

<0: `str1` is less than `str2` because  
char in `str1` is less than corresponding char in `str2`.

=0: `str1` is identical to `str2`

>0: `str1` is greater than `str2` because  
char in `str1` is greater than corresponding char in `str2`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strcmp`, `strcmpi`, `strncmpi`

---

---

## strncmpi

---

---

**NEAR SYNTAX:** `int _n_strncmpi( char * str1, char * str2, unsigned n );`  
**FAR SYNTAX:** `int _f_strncmpi( char far * str1, char far * str2, unsigned n );`

**Note:** By default, `strncmpi()` is defined to `_n_strncmpi()`.

### DESCRIPTION

Performs case-insensitive unsigned character by character comparison of two strings of length `n`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>str1</b>	Pointer to string 1.
<b>str2</b>	Pointer to string 2.
<b>n</b>	Maximum number of bytes to compare, if zero then strings are considered equal

### RETURN VALUE

`<0`: `str1` is less than `str2` because  
char in `str1` is less than corresponding char in `str2`.  
`=0`: `str1` is identical to `str2`  
`>0`: `str1` is greater than `str2` because  
char in `str1` is greater than corresponding char in `str2`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strcmpi`, `strcmp`, `strncmp`



---

---

## strncpy

---

---

**NEAR SYNTAX:** `char *_n_strncpy( char *dst, char *src, unsigned int n );`  
**FAR SYNTAX:** `char far * _f_strncpy( char far * dst, char far * src, size_t n );`

**Note:** By default, `strncpy()` is defined to `_n_strncpy()`.

### DESCRIPTION

Copies a given number of characters from one string to another and padding with null characters or truncating as necessary.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>dst</b>	Pointer to location to receive string.
<b>src</b>	Pointer to location to supply string.
<b>n</b>	Maximum number of bytes to copy. If equal to zero, this function has no effect.

### RETURN VALUE

Pointer to destination string.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strcpy`

---

---

## strpbrk

---

---

**NEAR SYNTAX:** `char * _n_strpbrk( char * s1, char * s2 );`

**FAR SYNTAX:** `char far * _f_strpbrk( char far * s1, char far * s2 );`

**Note:** By default, `strpbrk()` is defined to `_n_strpbrk()`.

### DESCRIPTION

Scans a string for the first occurrence of any character from another string.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<b>s1</b>	String to be scanned.
<b>s2</b>	Character occurrence string.

### RETURN VALUE

Pointer pointing to the first occurrence of a character contained in `s2` in `s1`. Returns null if not found.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strchr`, `strrchr`, `strtok`

---

---

## strrchr

---

---

**NEAR SYNTAX:** `char * _n_strrchr( char * s, int c );`

**FAR SYNTAX:** `char far * _f_strrchr( char far * s, int c );`

**Note:** By default, `strrchr()` is defined to `_n_strrchr()`.

### DESCRIPTION

Similar to `strchr`, except this function searches backward from the end of `s` to the beginning.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<code>s</code>	String to be searched
<code>c</code>	Search character

### RETURN VALUE

Pointer to last occurrence of `c` in `s`. If `c` is not found in `s`, return null.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strchr`, `strcspn`, `strtok`

---

---

## strspn

---

---

**NEAR SYNTAX:** `size_t _n_strspn( char * src, char * brk );`

**FAR SYNTAX:** `size_t _f_strspn( char far * src, char far * brk );`

**Note:** By default, `strspn()` is defined to `_n_strspn()`.

### DESCRIPTION

Scans a string for the first segment in `src` containing only characters specified in `brk`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

**src**                      String to be scanned

**brk**                      Set of characters

### RETURN VALUE

Returns the length of the segment.

### LIBRARY

`STRING.LIB`

---

---

## strstr

---

---

**NEAR SYNTAX:** `char * _n_strstr( char *s1, char *s2 );`

**FAR SYNTAX:** `char far * _f_strstr( char far * s1, char far * s2 );`

**Note:** By default, `strstr()` is defined to `_n_strstr()`.

### DESCRIPTION

Finds a substring specified by `s2` in string `s1`.

For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

### PARAMETERS

<code>s1</code>	String to be scanned.
<code>s2</code>	Substring to search for.

### RETURN VALUE

Pointer to the first occurrence of substring `s2` in `s1`. Returns null if `s2` is not found in `s1`.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strcspn`, `strchr`, `strtok`

---

---

## strtod

---

---

NEAR SYNTAX: `float _n_strtod( char * s, char ** tailptr );`

FAR SYNTAX: `float _f_strtod( char far * s, char far * far * tailptr );`

**Note:** By default, `strtod()` is defined to `_n_strtod()`.

### DESCRIPTION

ANSI string to float conversion.

For Rabbit 4000+ users, this function supports FAR pointers. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions by default. The user may also explicitly call the far version with `_f_strfunc`, where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g. `_n_strtod`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

**Warning:** The far version of `strtod` is **not** backwards compatible with near pointers due to the use of a double pointer. The problem is that `char ** tailptr` is a 16-bit pointer pointing to another 16-bit pointer. The far version, `char far * far * tailptr`, is a 32-bit pointer pointing to a 32-bit pointer. If you pass a double near pointer as the argument to the double far pointer function, the double dereference (`**tailptr`) of the double pointer will attempt to access a 32-bit address pointed to by the passed near pointer. The compiler does not know the contents of a pointer and will assume the inner pointer is a 32-bit pointer. For more information about FAR pointers, please see the *Dynamic C User's Manual*.

---

---

## strtod (cont'd)

---

---

In the following examples:

```
[ ] = 1 byte
[ ][ ][x][x] indicates a NEAR address (16 bit) upcast to FAR
```

Passing a “char far \* far \* ptr” as tailptr:

```
ADDRESS:          DATA:
[ ][ ][x][x]     [y][y][y][y] (tailptr)
[y][y][y][y]     [z][z][z][z] (*tailptr)
[z][z][z][z]     [Correct contents] (**tailptr)
```

Passing a 'char \*\* ptr' as tailptr: Note the first pointer can be upcast to FAR but the compiler doesn't know to upcast the internal pointer.

```
ADDRESS:          DATA:
[ ][ ][x][x]     [ ][ ][y][y] (tailptr)
[ ][ ][y][y]     [?][?][z][z] (*tailptr)
[?][?][z][z]     [Incorrect contents] (**tailptr)
```

### PARAMETERS

<b>s</b>	String to convert.
<b>tailptr</b>	Pointer to a pointer of character. The next conversion may resume at the location specified by *tailptr.

### RETURN VALUE

The float number represented by “s.”

### LIBRARY

STRING.LIB

### SEE ALSO

atof

---

---

## strtok

---

---

**NEAR SYNTAX:** `char * _n_strtok( char * src, char * brk );`

**FAR SYNTAX:** `char far * _f_strtok( char far * src, char far * brk );`

**Note:** By default, `strtok()` is defined to `_n_strtok()`.

### DESCRIPTION

Scans `src` for tokens separated by delimiter characters specified in `brk`.

First call with non-null for `src`. Subsequent calls with null for `src` continue to search tokens in the string. If a token is found (i.e., delimiters found), replace the first delimiter in `src` with a null terminator so that `src` points to a proper null terminated token.

### PARAMETERS

**src**                      String to be scanned, must be in SRAM, cannot be a constant. In contrast, strings initialized when they are declared are stored in flash memory, and are treated as constants.

**brk**                      Character delimiter.

### RETURN VALUE

Pointer to a token. If no delimiter (therefore no token) is found, returns null.

### LIBRARY

`STRING.LIB`

### SEE ALSO

`strchr`, `strrchr`, `strstr`, `strcspn`



---

---

## strtol

---

---

**NEAR SYNTAX:** `long _n_strtol( char * sptr, char ** tailptr, int base );`  
**FAR SYNTAX:** `long _f_strtol( char far *sptr, char far * far * tailptr,  
int base );`

**Note:** By default, `strtol()` is defined to `_n_strtol()`.

### DESCRIPTION

ANSI string to long conversion.

For Rabbit 4000+ users, this function supports FAR pointers. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions by default. The user may also explicitly call the far version with `_f_strfunc`, where `strfunc` is the name of the string function.

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g. `_n_strtod`. For more information about FAR pointers, see the *Dynamic C User's Manual* or the samples in `Samples/Rabbit4000/FAR/`.

**Warning:** The far version of `strtod` is **not** backwards compatible with near pointers due to the use of a double pointer. The problem is that `char ** tailptr` is a 16-bit pointer pointing to another 16-bit pointer. The far version, `char far * far * tailptr`, is a 32-bit pointer pointing to a 32-bit pointer. If you pass a double near pointer as the argument to the double far pointer function, the double dereference (`**tailptr`) of the double pointer will attempt to access a 32-bit address pointed to by the passed near pointer. The compiler does not know the contents of a pointer and will assume the inner pointer is a 32-bit pointer. For more information about FAR pointers, please see the *Dynamic C User's Manual*.

In the following examples:

```
[ ] = 1 byte  
[ ] [ ] [x] [x] indicates a NEAR address (16 bit) upcast to FAR
```

Passing a "char far \* far \* ptr" as `tailptr`:

```
ADDRESS:          DATA:  
[ ] [ ] [x] [x]   [y] [y] [y] [y] (tailptr)  
[y] [y] [y] [y]   [z] [z] [z] [z] (*tailptr)  
[z] [z] [z] [z]   [Correct contents] (**tailptr)
```

---

---

## strtol (cont'd)

---

---

Passing a 'char \*\* ptr' as tailptr: Note the first pointer can be upcast to FAR but the compiler doesn't know to upcast the internal pointer.

ADDRESS:	DATA:
[ ][ ][x][x]	[ ][ ][y][y] (tailptr)
[ ][ ][y][y]	[?][?][z][z] (*tailptr)
[?][?][z][z]	[Incorrect contents] (**tailptr)

### PARAMETERS

<b>sptr</b>	String to convert.
<b>tailptr</b>	Assigned the last position of the conversion. The next conversion may resume at the location specified by *tailptr.
<b>base</b>	Indicates the radix of conversion.

### RETURN VALUE

The long integer.

### LIBRARY

STRING.LIB

### SEE ALSO

atoi, atol

---

---

## `_sysIsSoftReset`

---

---

```
void _sysIsSoftReset ( void );
```

### DESCRIPTION

This function should be called at the start of a program if you are using protected variables. It determines whether this restart of the board is due to a software reset from Dynamic C or a call to `forceSoftReset()`. If it was a soft reset, this function then does the following:

- Calls `_prot_init()` to initialize the protected variable mechanisms. It is up to the user to initialize protected variables.
- Calls `sysResetChain()`. The user may attach functions to this chain to perform additional startup actions (for example, initializing protected variables). If a soft reset did not take place, this function calls `_prot_recover()` to recover any protected variables.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`chkHardReset`, `chkSoftReset`, `chkWDTO`

---

---

## `sysResetChain`

---

---

```
void sysResetChain ( void );
```

### DESCRIPTION

This is a function chain that should be used to initialize protected variables. By default, it's empty.

### LIBRARY

`SYS.LIB`

---

---

## tan

---

---

```
float tan ( float x );
```

### DESCRIPTION

Compute the tangent of the argument.

**Note:** The Dynamic C functions `deg()` and `rad()` convert radians and degrees.

### PARAMETERS

**x**                      Angle in radians.

### RETURN VALUE

Returns the tangent of `x`, where  $-8 \times \text{PI} \leq x \leq +8 \times \text{PI}$ . If `x` is out of bounds, the function returns 0 and signals a domain error. If the value of `x` is too close to a multiple of  $90^\circ$  ( $\text{PI}/2$ ) the function returns INF and signals a range error.

### LIBRARY

MATH.LIB

### SEE ALSO

`atan`, `cos`, `sin`, `tanh`

---

---

## tanh

---

---

```
float tanh ( float x );
```

### DESCRIPTION

Computes the hyperbolic tangent of argument. This functions takes a unitless number as a parameter and returns a unitless number.

### PARAMETERS

**x**                      Float to use in computation.

### RETURN VALUE

Returns the hyperbolic tangent of  $x$ . If  $x > 49.9$  (approx.), the function returns INF and signals a range error. If  $x < -49.9$  (approx.), the function returns -INF and signals a range error.

### LIBRARY

MATH.LIB

### SEE ALSO

atan, cosh, sinh, tan

---

---

## TAT1R\_SetValue

---

---

```
char TAT1R_SetValue( int requestor, int value );
```

### DESCRIPTION

If not already in use, or if in a compatible use, allocates the TAT1R resource (sets a new or keeps the current TAT1R value) as requested. Also enables or disables the requestor's timer A1 cascade bit(s) in TACR or TBCR, as appropriate. When the timer B cascade from timer A1 is disabled in TBCR the timer B "clocked by PCLK/2" is then enabled.

A run time error occurs if parameter(s) are invalid and also, this function is not reentrant.

**Note:** This function does not attempt to manage interrupts that are associated with timers A or B; that work is left entirely up to the application.

### PARAMETERS

<b>requestor</b>	The requestor of the TAT1R resource. Use exactly one of the following macros to specify the appropriate requestor: <ul style="list-style-type: none"><li>• TAT1R_A1TIMER_REQ (e.g., direct use of Timer A1)</li><li>• TAT1R_A2TIMER_REQ (e.g., use by serial port E)</li><li>• TAT1R_A3TIMER_REQ (e.g., use by serial port F)</li><li>• TAT1R_A4TIMER_REQ (e.g., use by serial port A)</li><li>• TAT1R_A5TIMER_REQ (e.g., use by serial port B)</li><li>• TAT1R_A6TIMER_REQ (e.g., use by serial port C)</li><li>• TAT1R_A7TIMER_REQ (e.g., use by serial port D)</li><li>• TAT1R_BTIMER_REQ (e.g., use with PWM, servo or triac)</li></ul>
<b>value</b>	Either the new TAT1R setting value (0 to 255, inclusive), or the macro TAT1R_RELEASE_REQ to release the TAT1R resource in use by the specified requestor.

### RETURN VALUE

The new or current TAT1R setting. The caller should check their requested new TAT1R value against this return value. If the two values are not the same, the caller may decide the return value is acceptable after all and make another request using the previous return value. A valid release request always succeeds; in this case there is no need for the caller to check the return value.

### LIBRARY

sys.lib

---

---

## tm\_rd

---

---

```
int tm_rd( struct tm * t );
```

### DESCRIPTION

Reads the current system time from SEC\_TIMER into the structure t.

WARNING: The variable SEC\_TIMER is initialized when a program is started. If you change the Real Time Clock (RTC), this variable will not be updated until you restart a program, and the tm\_rd() function will not return the time that the RTC has been reset to. The read\_rtc() function will read the actual RTC and can be used if necessary.

### PARAMETERS

**t**                      Pointer to structure to store time and date.

```
struct tm {
    char tm_sec;      // seconds 0-59
    char tm_min;     // 0-59
    char tm_hour;    // 0-23
    char tm_mday;    // 1-31
    char tm_mon;     // 1-12
    char tm_year;    // 80-147 (1980-2047)
    char tm_wday;    // 0-6 0==Sunday
};
```

### RETURN VALUE

0: Successful.  
-1: Clock read failed.

### LIBRARY

RTCLOCK.LIB

### SEE ALSO

mktm, mktime, tm\_wr

---

---

## tm\_wr

---

---

```
int tm_wr( struct tm * t );
```

### DESCRIPTION

Sets the system time from a `tm` struct. It is important to note that although `tm_rd()` reads the `SEC_TIMER` variable, not the RTC, `tm_wr()` writes to the RTC directly, and `SEC_TIMER` is not changed until the program is restarted. The reason for this is so that the `DelaySec()` function continues to work correctly after setting the system time. To make `tm_rd()` match the new time written to the RTC without restarting the program, the following should be done:

```
tm_wr(tm);  
SEC_TIMER = mktime(tm);
```

But this could cause problems if a `waitFor(DelaySec(n))` is pending completion in a co-operative multitasking program or if the `SEC_TIMER` variable is being used in another way the user, so user beware.

### PARAMETERS

<b>t</b>	Pointer to structure to read date and time from.
----------	--

```
struct tm {  
    char tm_sec;      // seconds 0-59  
    char tm_min;      // 0-59  
    char tm_hour;     // 0-23  
    char tm_mday;     // 1-31  
    char tm_mon;      // 1-12  
    char tm_year;     // 80-147 (1980-2047)  
    char tm_wday;     // 0-6 0==Sunday  
};
```

### RETURN VALUE

0: Success .  
-1: Failure.

### LIBRARY

RTCLOCK.LIB

### SEE ALSO

`mktm`, `mktime`, `tm_rd`



---

---

## tolower

---

---

```
int tolower( int c );
```

### DESCRIPTION

Convert alphabetic character to lower case.

### PARAMETERS

**c**                      Character to convert

### RETURN VALUE

Lower case alphabetic character.

### LIBRARY

STRING.LIB

### SEE ALSO

toupper, isupper, islower

---

---

## toupper

---

---

```
int toupper( int c );
```

### DESCRIPTION

Convert alphabetic character to uppercase.

### PARAMETERS

**c**                      Character to convert.

### RETURN VALUE

Upper case alphabetic character.

### LIBRARY

STRING.LIB

### SEE ALSO

tolower, isupper, islower

---

---

## updateTimers

---

---

```
void updateTimers( void );
```

### DESCRIPTION

Updates the values of `TICK_TIMER`, `MS_TIMER`, and `SEC_TIMER` while running off the 32 kHz oscillator. Since the periodic interrupt is disabled when running at 32 kHz, these values will not be updated unless this function is called.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`useMainOsc`, `use32kHzOsc`

---

---

## use32kHzOsc

---

---

```
void use32kHzOsc( void );
```

### DESCRIPTION

Sets the Rabbit processor to use the 32kHz real-time clock oscillator for both the CPU and peripheral clock, and shuts off the main oscillator. If this is already set, there is no effect. This mode should provide greatly reduced power consumption. Serial communications will be lost since typical baud rates cannot be made from a 32kHz clock. Also note that this function disables the periodic interrupt, so `waitfor` and related statements will not work properly (although costatements in general will still work). In addition, the values in `TICK_TIMER`, `MS_TIMER`, and `SEC_TIMER` will not be updated unless you call the function `updateTimers()` frequently in your code. In addition, you will need to call `hitwd()` periodically to hit the hardware watchdog timer since the periodic interrupt normally handles that, or disable the watchdog timer before calling this function. The watchdog can be disabled with `Disable_HW_WDT()`.

`use32kHzOsc()` is not task reentrant.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`useMainOsc`, `useClockDivider`, `updateTimers`

---

---

## useClockDivider

---

---

```
void useClockDivider( void );
```

### DESCRIPTION

Sets the Rabbit processor to use the main oscillator divided by 8 for the CPU (but not the peripheral clock). If this is already set, there is no effect. Because the peripheral clock is not affected, serial communications should still work. This function also enables the periodic interrupt in case it was disabled by a call to `use32kHzOsc()`.

This function is not task reentrant.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`useMainOsc`, `use32kHzOsc`

---

---

## useClockDivider3000

---

---

```
void useClockDivider3000( int setting );
```

### DESCRIPTION

Sets the expanded clock divider options for the Rabbit 3000 processor. Target communications will be lost after changing this setting because of the baud rate change. This function also enables the periodic interrupt in case it was disabled by a call to `user32kHzOsc()`.

The peripheral clock is also affected by this function. If you want to divide the main processor clock and not the peripheral clock, you may use the function `useClockDivider()` to divide the main processor clock by 8. To divide the main processor clock by any of the other allowable values (2, 4, or 6) means using `useClockDivider3000()` and thus dividing the peripheral clock as well.

This function is not task reentrant.

### PARAMETER

<b>setting</b>	Divider setting. The following are valid: <ul style="list-style-type: none"><li>• CLKDIV_2 - divide main processor clock by two</li><li>• CLKDIV_4 - divide main processor clock by four</li><li>• CLKDIV_6 - divide main processor clock by six</li><li>• CLKDIV_8 - divide main processor clock by eight</li></ul>
----------------	--

### RETURN VALUE

None.

### LIBRARY

SYS.LIB

### SEE ALSO

`useClockDivider`, `useMainOsc`, `use32kHzOsc`, `set32kHzDivider`

---

---

## useMainOsc

---

---

```
void useMainOsc( void );
```

### DESCRIPTION

Sets the Rabbit processor to use the main oscillator for both the CPU and peripheral clock. If this is already set, there is no effect. This function also enables the periodic interrupt in case it was disabled by a call to `use32kHzOsc()`, and updates the `TICK_TIMER`, `MS_TIMER`, and `SEC_TIMER` variables from the real-time clock. This function is not task reentrant.

### LIBRARY

`SYS.LIB`

### SEE ALSO

`use32kHzOsc`, `useClockDivider`

---

---

## utoa

---

---

```
char * utoa( unsigned value, char * buf );
```

### DESCRIPTION

Places up to 5 digit character string at `*buf` representing value of unsigned number. Suppresses leading zeros, but leaves one zero digit for value = 0. Max = 65535. 73 program bytes.

### PARAMETERS

<b>value</b>	16-bit number to convert.
<b>buf</b>	Character string of converted number.

### RETURN VALUE

Pointer to null at end of string.

### LIBRARY

`STDIO.LIB`

### SEE ALSO

`itoa`, `htoa`, `ltoa`

---

---

## vram2root

---

---

```
int vram2root( void * dest, int start, int length );
```

### DESCRIPTION

This function copies data from the VBAT RAM. Tamper detection on the Rabbit 4000 erases the VBAT RAM with any attempt to enter bootstrap mode.

### PARAMETERS

<b>dest</b>	The address to which the data in the VBAT RAM will be copied.
<b>start</b>	The start location within the VBAT RAM (0-31).
<b>length</b>	The length of data to read from VBAT RAM. The length should be greater than 0. The parameters <code>length + start</code> should not exceed 32.

### LIBRARY

VBAT.LIB

### SEE ALSO

`root2vram`

---

---

## VdGetFreeWd

---

---

```
int VdGetFreeWd( char count );
```

### DESCRIPTION

Returns a free virtual watchdog and initializes that watchdog so that the virtual driver begins counting it down from `count`. The number of available virtual watchdogs is determined by the macro `N_WATCHDOG`, which is 10 by default. The default can be overridden by the user, e.g., `#define N_WATCHDOG 11`.

The virtual driver is called every 0.00048828125 second. On every 128th call to it (i.e., every 62.5 ms), the virtual watchdogs are counted down and then tested. If any virtual watchdog reaches zero, this is a fatal error. Once a virtual watchdog is active, it should reset periodically with a call to `VdHitWd()` to prevent the count from reaching zero.

### PARAMETERS

`count`             $1 < \text{count} \leq 255$

### RETURN VALUE

Integer id number of an unused virtual watchdog timer.

### LIBRARY

`VDRIVER.LIB`

---

---

## VdHitWd

---

---

```
int VdHitWd( int ndog );
```

### DESCRIPTION

Resets virtual watchdog counter to N counts where N is the argument to the call to `VdGetFreeWd()` that obtained the virtual watchdog `ndog`.

The virtual driver counts down watchdogs every 62.5 ms. If a virtual watchdog reaches 0, this is a fatal error. Once a virtual watchdog is active it should reset periodically with a call to `VdHitWd()` to prevent this.

If  $N = 2$ , `VdHitWd()` will need to be called again for virtual watchdog `ndog` within 62.5 ms.

If  $N = 255$ , `VdHitWd()` will need to be called again for virtual watchdog `ndog` within 15.9375 seconds.

### PARAMETERS

**ndog**                    Id of virtual watchdog returned by `VdGetFreeWd()`

### LIBRARY

VDRIVER.LIB

---

---

## VdInit

---

---

```
void VdInit( void );
```

### DESCRIPTION

Initializes the Virtual Driver for all Rabbit boards. Supports `DelayMs()`, `DelaySec()`, `DelayTick()`. `VdInit()` is called by the BIOS unless it has been disabled.

### LIBRARY

VDRIVER.LIB



---

---

## VdReleaseWd

---

---

```
int VdReleaseWd( int ndog );
```

### DESCRIPTION

Deactivates a virtual watchdog and makes it available for `VdGetFreeWd()`.

### PARAMETERS

**ndog**                    Handle returned by `VdGetFreeWd()`

### RETURN VALUE

0: ndog out of range.  
1: Success.

### LIBRARY

VDRIVER.LIB

### EXAMPLE

```
// VdReleaseWd virtual watchdog example
main() {
    int wd;                               // handle for a virtual watchdog
    unsigned long tm;
    tm = SEC_TIMER;
    wd = VdGetFreeWd(255);                 // wd activated, 9 virtual watchdogs
                                           // now available. wd must be hit
                                           // at least every 15.875 seconds

    while(SEC_TIMER - tm < 60) {         // let it run for a minute
        VdHitWd(wd);                       // reset counter back to 255
    }
    VdReleaseWd(wd)                         // now 10 virtual watchdogs available
}
```

---

---

## WriteFlash2

---

---

```
int WriteFlash2( unsigned long flashDst, void * rootSrc,
                unsigned len );
```

### DESCRIPTION

Write `len` bytes from `rootSrc` to physical address `flashDst` on the 2nd flash device. The source must be in root. The `flashDst` address plus the sum of `numbytes[]` area must be within memory quadrant(s) already mapped to the second flash.

This function is not reentrant.

**Note:** This function should NOT be used if you are using the second flash device for a flash file system, e.g. if you are writing a TCP/IP-based application!

**Note:** This function is extremely dangerous when used with large sector flash. Don't do it.

### PARAMETERS

<code>flashDst</code>	Physical address of the flash destination
<code>rootSrc</code>	Pointer to the root source
<code>len</code>	Number of bytes to write

### RETURN VALUE

- 0: Success.
- 1: Attempt to write non-2nd flash area, nothing written.
- 2: `rootSrc` not in root.
- 3: Time out while writing flash.
- 4: Attempt to write to ID block
- 5: Sector erase needed; write aborted

### LIBRARY

XMEM.LIB

---

---

## WriteFlash2Array

---

---

```
int WriteFlash2Array( unsigned long flashDst, void * rootSrc[],
    unsigned numbytes[], int numsources );
```

### DESCRIPTION

Write a set of scattered information to the 2nd flash in a contiguous block. The sources are given in the `rootSrc` array, and the corresponding number of bytes in each source is given in the `numbytes []` array. All sources must be in root. `numsources` specifies the number of entries in the `rootSrc` and `numbytes` arrays. The `flashDst` address plus the sum of `numbytes []` area must be within memory quadrant(s) already mapped to the second flash.

This function is not reentrant. It was introduced in Dynamic C version 7.30.

**Note:** This function should NOT be used if you are using the second flash device for a flash file system, e.g. if you are writing a TCP/IP-based application!

**Note:** This function is extremely dangerous when used with large sector flash. Don't do it.

**Note:** The sum of the lengths in `numbytes []` must not exceed 65535 bytes, else not all data will be written.

### PARAMETERS

<b>flashDst</b>	Physical address of the flash destination.
<b>rootSrc</b>	Array of pointers to the root sources.
<b>numbytes</b>	Array of numbers of bytes to write for each source.
<b>numsources</b>	Number of sources specified in <code>rootSrc []</code> and <code>numbytes []</code> .

### RETURN VALUE

0: Success.  
-1: Attempt to write non-2nd flash area, nothing written.  
-2: `rootSrc []` entry not in root.  
-3: Time-out while writing flash.

### LIBRARY

XMEM.LIB

---

---

## write\_rtc

---

---

```
void write_rtc( unsigned long int time );
```

### DESCRIPTION

Writes a 32 bit seconds value to the RTC, zeros other bits. This function does not stop or delay periodic interrupt. It does not affect the SEC\_TIMER or MS\_TIMER variables.

### PARAMETERS

<b>time</b>	32-bit value representing the number of seconds since January 1, 1980.
-------------	--

### LIBRARY

RTCLOCK.LIB

### SEE ALSO

read\_rtc

---

---

## writeUserBlock

---

---

```
int writeUserBlock( unsigned addr, void *source, unsigned numbytes );
```

### DESCRIPTION

Rabbit-based boards have a System ID block located on the primary flash. (See the *Rabbit Microprocessor Designer's Handbook* for more information on the System ID block.) Version 2 and later of this ID block has a pointer to a User ID block: a place intended for storing calibration constants, passwords, and other non-volatile data.

The User block is recommended for storing all non-file data. The User block is where calibration constants are stored for boards with analog I/O. Space in the User block is limited to as small as  $(8K - \text{sizeof}(\text{SysIDBlock}))$  bytes, or less, if there are calibration constants.

`writeUserBlock()` writes a number of bytes from root memory to the User block. This block is protected from normal writes to the flash device and can only be accessed through this function or the function `writeUserBlockArray()`.

Using this function can cause all interrupts to be disabled for as long as 20 ms while a flash sector erases, depending on the flash type. A single call can produce as many as four of these erase delays. This will cause periodic interrupts to be missed, and can cause other interrupts to be missed as well. Therefore, it is best to buffer up data to be written rather than to do many writes.

While debugging, several consecutive calls to this function can cause a loss of target serial communications. This effect can be reduced by introducing delays between the calls, lowering the baud rate, or increasing the serial time-out value in the project file.

**Note:** See the manual for your particular board for more information before overwriting any part of the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), `writeUserBlock()` should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using  $\mu\text{C}/\text{OS-II}$  and `_SPI_USE_UCOS_MUTEX` is `#defined`, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error `ERR_SPI_MUTEX_ERROR` will occur. See the description for `_rcm43_InitUCOSMutex()` for more information on using  $\mu\text{C}/\text{OS-II}$  and `_SPI_USE_UCOS_MUTEX`.

### Backwards Compatibility:

If the version of the System ID block doesn't support the User ID block, or no System ID block is present, then 8K bytes starting 16K bytes from the top of the primary flash are designated the User ID block area. However, to prevent errors arising from incompatible large sector configurations, this will only work if the flash type is small sector. Rabbit Semiconductor manufactured boards with large sector flash will have valid System and User ID blocks, so this should not be problem on Rabbit boards.

If users create boards with large sector flash, they must install System ID blocks version 2 or greater to use or modify this function.

---

---

## **writeUserBlock (cont'd)**

---

---

### **PARAMETERS**

**addr**                    Address offset in User block to write to.

**source**                 Pointer to source to copy data from.

**numbytes**              Number of bytes to copy.

### **RETURN VALUE**

0: Successful  
-1: Invalid address or range

The return values below are new with Dynamic C 10.21:

-2: No valid user block found (block version 3 or later)  
-3: flash writing error

The return values below are applicable only if `_SPI_USE_UCOS_MUTEX` is not #defined:

-ETIME: (Serial flash only, time out waiting for SPI)  
positive N: (Serial flash only, SPI in use by device N)

### **LIBRARY**

IDBLOCK.LIB

### **SEE ALSO**

`readUserBlock`, `writeUserBlockArray`

---

---

## writeUserBlockArray

---

---

```
int writeUserBlockArray( unsigned addr, void * sources[], unsigned
    numbytes[], int numsources );
```

### DESCRIPTION

Rabbit Semiconductor boards are released with System ID blocks located on the primary flash. Version 2 and later of this ID block has a pointer to a User block that can be used for storing calibration constants, passwords, and other non-volatile data. The User block is protected from normal write to the flash device and can only be accessed through this function or `writeUserBlock()`.

This function writes a set of scattered data from root memory to the User block. If the data to be written are in contiguous bytes, using the function `writeUserBlock()` is sufficient. Use of `writeUserBlockArray()` is recommended when the data to be written is in noncontiguous bytes, as may be the case for something like network configuration data.

See the *Rabbit Microprocessor Designer's Handbook* for more information about the System ID and User blocks.

**Note:** Portions of the User block may be used by the BIOS for your board to store values, e.g., calibration constants. See the manual for your particular board for more information before overwriting any part of the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), `writeUserBlockArray()` should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using  $\mu$ C/OS-II and `_SPI_USE_UCOS_MUTEX` is `#defined`, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error `ERR_SPI_MUTEX_ERROR` will occur. See the description for `_rcm43_InitUCOSMutex()` for more information on using  $\mu$ C/OS-II and `_SPI_USE_UCOS_MUTEX`.

### Backwards Compatibility:

If the System ID block on the board doesn't support the User block, or no System ID block is present, then the 8K bytes starting 16K bytes from the top of the primary flash are designated User block area. This only works if the flash type is small sector. Rabbit manufactured boards with large sector flash will have valid System ID and User blocks, so is not a problem on Rabbit boards. If users create boards with large sector flash, they must install System ID blocks version 3 or greater to use this function, or modify this function.

---

---

## **writeUserBlockArray**

---

---

### **PARAMETERS**

<b>addr</b>	Address offset in User block to write to.
<b>sources</b>	Array of pointer to sources to copy data from.
<b>numbytes</b>	Array of number of bytes to copy for each source. The sum of the lengths in this array must not exceed 32767 bytes, or an error will be returned.
<b>numsources</b>	Number of data sources.

### **RETURN VALUE**

- 0: Successful.
- 1: Invalid address or range.
- 2: No valid User block found (block version 3 or later).
- 3: Flash writing error.

The return values below are applicable only if `_SPI_USE_UCOS_MUTEX` is not `#defined`:

- ETIME: (Serial flash only, time out waiting for SPI)
- postive N: (Serial flash only, SPI in use by device N)

### **LIBRARY**

IDBLOCK.LIB



---

---

## WrPortE

---

---

```
void WrPortE( unsigned int port, char * portshadow, int data_value);
```

### DESCRIPTION

Writes an external I/O register with 8 bits and updates shadow for that register. The variable names must be of the form `port` and `portshadow` for the most efficient operation. A null pointer may be substituted if shadow support is not desired or needed.

### PARAMETERS

<code>port</code>	Address of external data register.
<code>portshadow</code>	Reference pointer to a variable shadowing the register data. Substitute with null pointer (or 0) if shadowing is not required.
<code>data_value</code>	Value to be written to the data register

### LIBRARY

`SYSIO.LIB`

### SEE ALSO

`RdPortI`, `BitRdPortI`, `WrPortI`, `BitWrPortI`, `RdPortE`, `BitRdPortE`,  
`BitWrPortE`

---

---

## WrPortI

---

---

```
void WrPortI( int port, char * portshadow, int data_value );
```

### DESCRIPTION

Writes an internal I/O register with 8 bits and updates shadow for that register.

### PARAMETERS

<b>port</b>	Address of data register.
<b>portshadow</b>	Reference pointer to a variable shadowing the register data. Substitute with null pointer (or 0) if shadowing is not required.
<b>data_value</b>	Value to be written to the data register

### LIBRARY

SYSIO.LIB

### SEE ALSO

RdPortI, BitRdPortI, BitRdPortE, BitWrPortI, RdPortE, WrPortE,  
BitWrPortE

---

---

## xalloc

---

---

```
long xalloc( long sz );
```

### DESCRIPTION

Allocates the specified number of bytes in extended memory. Starting with Dynamic C version 7.04P3, the returned address is always even (word) aligned.

Starting with Dynamic C 8, if `xalloc()` fails, a run-time error will occur. This is a wrapper function for `_xalloc()`, for backwards compatibility. It is the same as `_xalloc(&sz, 1, XALLOC_MAYBBB)` except that the actual allocated amount is not returned since the parameter is not a pointer.

Starting with Dynamic C 9.30, `xalloc()` and related functions were modified so that they are now driven by the compiler origin directives.

**Note:** `xalloc()` is not thread safe since it accesses a global static structure with no locking.

### PARAMETERS

**sz**                      Number of bytes to allocate. This is rounded up to the next higher even number.

### RETURN VALUE

The 20-bit physical address of the allocated data: Success.  
0: Failure.

**Note:** Starting with Dynamic C 8, a run-time exception will occur if the function fails.

### LIBRARY

STACK.LIB

### SEE ALSO

`root2xmem`, `xmem2root`, `xavail`

---

---

## \_xalloc

---

---

```
long _xalloc( long * sz, word align, word type );
```

### DESCRIPTION

Allocates memory in extended memory. If `_xalloc()` fails, a runtime error will occur.

### PARAMETERS

- |              |   |
|--------------|---|
| <b>sz</b>    | On entry, pointer to the number of bytes to allocate. On return, the pointed-to value will be updated with the actual number of bytes allocated. This may be larger than requested if an odd number of bytes was requested, or if some space was wasted at the end because of alignment restrictions.   |
| <b>align</b> | Storage alignment as the log (base 2) of the desired returned memory starting address. For example, if this parameter is "8," then the returned address will align on a 256-byte boundary. Values between 0 and 16 inclusive are allowed. Any other value is treated as zero, i.e., no required alignment.  |
| <b>type</b>  | This parameter is only meaningful on boards with more than one type of RAM. For example, boards with a fast RAM and a slower battery-backed RAM like the RCM3200 or RCM3300 Use one of the following values, any other value will have undefined results. <ul style="list-style-type: none"><li>• <code>XALLOC_ANY</code> (0) - any type of SRAM storage allowed</li><li>• <code>XALLOC_BB</code> (1) - must be battery-backed program execution SRAM (a.k.a., fast RAM).</li><li>• <code>XALLOC_NOTBB</code> (2) - return non-BB SRAM only.</li><li>• <code>XALLOC_MAYBBB</code> (3) - return non-BB SRAM in preference to BB.</li></ul> |

### RETURN VALUE

The 20-bit physical address of the allocated data on success. On error, a runtime error occurs.

**Note:** This return value cannot be used with pointer arithmetic.

### LIBRARY

`STACK.LIB`

### EXCEPTIONS

`ERR_BADXALLOC` - if could not allocate requested storage, or negative size passed.

---

---

## `xalloc_stats`

---

---

```
void xalloc_stats( word parm );
```

### DESCRIPTION

Prints a table of available `xalloc()` regions to the Stdio window.

This function was introduced in Dynamic C version 8. It is for debugging and educational purposes. It should not be called in a production program.

### PARAMETERS

**parm**                      Prior to Dynamic C version 9.30: reserved for future use. Set to 0.  
Starting with DC 9.30: this parameter is of type long.

### LIBRARY

MEM.LIB (XMEM.LIB prior to DC 9.30)

### SEE ALSO

`xalloc`, `_xalloc`, `xavail`, `_xavail`, `xrelease`

---

---

## xavail

---

---

```
long xavail( long * addr_ptr );
```

### DESCRIPTION

Returns the maximum length of memory that may be successfully obtained by an immediate call to `xalloc()`, and optionally allocates that amount.

This function was introduced in Dynamic C version 7.04P3.

### PARAMETERS

**addr\_ptr**            Pointer to a long word in root data memory to store the address of the block. If this pointer is null, then the block is not allocated. Otherwise, the block is allocated as if by a call to `xalloc()`.

### RETURN VALUE

The size of the largest free block available. If this is zero, then `*addr_ptr` will not be changed.

### LIBRARY

XMEM.LIB (was in STACK.LIB prior to DC 8)

### SEE ALSO

`xalloc`, `_xalloc`, `_xavail`, `xrelease`, `xalloc_stats`

---

---

## **`_xavail`**

---

---

```
long _xavail( long * addr_ptr, word align, word type );
```

### **DESCRIPTION**

Returns the maximum length of memory that may be successfully obtained by an immediate call to `_xalloc()`, and optionally allocates that amount. The `align` and `type` parameters are the same as would be presented to `_xalloc()`.

### **PARAMETERS**

<b><code>addr_ptr</code></b>	Address of a longword, in root data memory, to store the address of the block. If this pointer is null, then the block is not allocated. Otherwise, the block is allocated as if by a call to <code>_xalloc()</code> .
<b><code>align</code></b>	Alignment of returned block, as per <code>_xalloc()</code> .
<b><code>type</code></b>	Type of memory, as per <code>_xalloc()</code> .

### **RETURN VALUE**

The size of the largest free block available. If this is zero, then `*addr_ptr` will not be changed.

### **LIBRARY**

`XMEM.LIB`

### **SEE ALSO**

`xalloc`, `_xalloc`, `xavail`, `xrelease`, `xalloc_stats`

---

---

## **xCalculateECC256**

---

---

```
long xCalculateECC256( unsigned long data );
```

### **DESCRIPTION**

Calculates a 3 byte Error Correcting Checksum (ECC, 1 bit correction and 2 bit detection capability) value for a 256 byte (2048 bit) data buffer located in extended memory.

### **PARAMETERS**

**data**                      Physical address of the 256 byte data buffer.

### **RETURN VALUE**

The calculated ECC in the 3 LSBs of the long (i.e., BCDE) result. Note that the MSB (i.e., B) of the long result is always zero.

### **LIBRARY**

ECC.LIB (This function was introduced in Dynamic C 9.01)



---

---

## xChkCorrectECC256

---

---

```
int xChkCorrectECC256( unsigned long data, void * old_ecc,  
    void * new_ecc );
```

### DESCRIPTION

Checks the old versus new ECC values for a 256 byte (2048 bit) data buffer, and if necessary and possible (1 bit correction, 2 bit detection), corrects the data in the specified extended memory buffer.

### PARAMETERS

<b>data</b>	Physical address of the 256 byte data buffer
<b>old_ecc</b>	Pointer to the old (original) 3 byte ECC's buffer
<b>new_ecc</b>	Pointer to the new (current) 3 byte ECC's buffer

### RETURN VALUE

- 0: Data and ECC are good (no correction is necessary)
- 1: Data is corrected and ECC is good
- 2: Data is good and ECC is corrected
- 3: Data and/or ECC are bad and uncorrectable

### LIBRARY

ECC.LIB (This function was introduced in Dynamic C 9.01)

---

---

## xgetfloat

---

---

```
float xgetfloat( long src );
```

### DESCRIPTION

Returns the `float` pointed to by `src`. This is the most efficient function for obtaining 4 bytes from `xmem`.

### PARAMETERS

**src**                    `xmem` (linear) address of the float value to retrieve.

### RETURN VALUE

`float` value (4 bytes) at `src`.

### LIBRARY

`XMEM.LIB`

---

---

## xgetint

---

---

```
int xgetint( long src );
```

### DESCRIPTION

Returns the integer pointed to by `src`. This is the most efficient function for obtaining 2 bytes from `xmem`.

### PARAMETERS

**src**                    `xmem` (linear) address of the integer value to retrieve.

### RETURN VALUE

Integer value (2-bytes) at `src`.

### LIBRARY

`XMEM.LIB`

---

---

## **xgetlong**

---

---

```
long xgetlong( long src );
```

### **DESCRIPTION**

Return the long word pointed to by `src`. This is the most efficient function for obtaining 4 bytes from `xmem`.

### **PARAMETERS**

`src`                    `xmem` (linear) address of the long value to retrieve.

### **RETURN VALUE**

Long integer value (4 bytes) at `src`.

### **LIBRARY**

`XMEM.LIB`

---

---

## xmem2root

---

---

```
int xmem2root( void * dest, unsigned long int src,
               unsigned int len );
```

### DESCRIPTION

Stores `len` characters from physical address `src` to logical address `dest`.

### PARAMETERS

<code>dest</code>	Logical address
<code>src</code>	Physical address
<code>len</code>	Numbers of bytes

### RETURN VALUE

0: Success.  
-1: Attempt to write flash memory area, nothing written.  
-2: Destination not all in root.

### LIBRARY

`XMEM.LIB`

### SEE ALSO

`root2xmem`, `xalloc`

---

---

## **xmem2xmem**

---

---

```
int xmem2xmem( unsigned long dest, unsigned long src,
              unsigned len );
```

### **DESCRIPTION**

Stores `len` characters from physical address `src` to physical address `dest`.

### **PARAMETERS**

<b>dest</b>	Physical address of destination
<b>src</b>	Physical address of source data
<b>len</b>	Length of source data in bytes

### **RETURN VALUE**

0: Success.  
-1: Attempt to write flash memory area, nothing written.

### **LIBRARY**

XMEM.LIB

---

---

## **xmemchr**

---

---

```
long xmemchr( long src, char ch, unsigned short n );
```

### **DESCRIPTION**

Search for the first occurrence of character `ch` in the `xmem` area pointed to by `src`.

### **PARAMETERS**

<b>src</b>	xmem (linear) address of the first character to search.
<b>ch</b>	Character to search for.
<b>n</b>	Maximum number of characters to search.

### **RETURN VALUE**

0: Character was not found within `n` bytes from the start.  
>0: Physical address of the first character that matched `ch`.

### **LIBRARY**

XMEM.LIB

---

---

## **xmemcmp**

---

---

```
int xmemcmp( long xstr, char * str, unsigned short n );
```

### **DESCRIPTION**

Test whether xmem string at `xstr` matches the root memory string at `str`. `n` bytes are compared.

### **PARAMETERS**

<b>xstr</b>	xmem (linear) address of the first character of the first string to compare.
<b>str</b>	root address of the first character of the second string to compare.
<b>n</b>	Length of each string. If <code>n</code> is zero, returns zero. <code>n</code> must be less than or equal 4097.

### **RETURN VALUE**

0: Exact match.  
>0: `xstr > str`  
<0: `xstr < str`

### **LIBRARY**

`XMEM.LIB`

---

---

## xrelease

---

---

```
void xrelease( long addr, long sz );
```

### DESCRIPTION

Release a block of memory previously obtained by `xalloc()` or by `xavail()` with a non-null parameter. `xrelease()` may only be called to free the most recent block obtained. It is NOT a general-purpose malloc/free type of dynamic memory allocation. Calls to `xalloc()/xrelease()` must be nested in first-allocated/last-released order, similar to the execution stack. The `addr` parameter must be the return value from `xalloc()`. If not, then a run-time exception will occur. The `sz` parameter must also be equal to the actual allocated size, however this is not checked. The actual allocated size may be larger than the requested size (because of alignment overhead). The actual size may be obtained by calling `_xalloc()` rather than `xalloc()`. For this reason, it is recommended that your application consistently uses `_xalloc()` rather than `xalloc()` if you intend to use this function.

### PARAMETERS

<b>addr</b>	Address of storage previously obtained by <code>_xalloc()</code> .
<b>sz</b>	Size of storage previously returned by <code>_xalloc()</code> .

### LIBRARY

XMEM.LIB

### SEE ALSO

`xalloc`, `_xalloc`, `xavail`, `_xavail`, `xalloc_stats`



---

---

## **xsetint**

---

---

```
void xsetint( long dst, int val );
```

### **DESCRIPTION**

Set the integer pointed to by `dst`. This is the most efficient function for writing two bytes to `xmem`.

### **PARAMETERS**

<b>dst</b>	xmem (linear) address of the int value to set.
<b>val</b>	value to store into the above location.

### **RETURN VALUE**

None

### **LIBRARY**

`XMEM.LIB`

---

---

## **xsetfloat**

---

---

```
void xsetfloat( long dst, float val );
```

### **DESCRIPTION**

Set the float pointed to by `dst`. This is the most efficient function for writing 4 bytes to `xmem`.

### **PARAMETERS**

<b>dst</b>	xmem (linear) address of the float value to set.
<b>val</b>	value to store into the above location.

### **RETURN VALUE**

None

### **LIBRARY**

`XMEM.LIB`

---

---

## xsetlong

---

---

```
void xsetlong( long dst, long val );
```

### DESCRIPTION

Set the long integer pointed to by `dst`. This is the most efficient function for writing 4 bytes to `xmem`.

### PARAMETERS

<b>dst</b>	xmem (linear) address of the long integer value to set.
<b>val</b>	value to store into the above location.

### RETURN VALUE

None

### LIBRARY

`XMEM.LIB`

---

---

## xstrlen

---

---

```
unsigned int xstrlen( long src );
```

### DESCRIPTION

Return the length of the string in `xmem` pointed to by `src`. If there is no null terminator within the first 65536 bytes of the string, then the return value will be meaningless.

### PARAMETERS

<b>src</b>	xmem (linear) address of the first character of the string. Note: to perform a normal null-terminated search, ensure that <code>src</code> is in the range $0..2^{20}-1$ . If the MSB of <code>src</code> is not zero (i.e., bits 24-31) then that character will be used to terminate the search rather than the standard null terminator. E.g., to determine the length of a string terminated by '@':
------------	--

```
xstrlen( paddr(my_str) | (long) '@' << 24 );
```

### RETURN VALUE

Length of string, not counting the terminator.

### LIBRARY

`XMEM.LIB`

## Software License Agreement

# RABBIT® SOFTWARE END USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: BY INSTALLING, COPYING OR OTHERWISE USING THE ENCLOSED RABBIT DYNAMIC C SOFTWARE, WHICH INCLUDES COMPUTER SOFTWARE ("SOFTWARE") AND MAY INCLUDE ASSOCIATED MEDIA, PRINTED MATERIALS, AND "ONLINE" OR ELECTRONIC DOCUMENTATION ("DOCUMENTATION"), YOU (ON BEHALF OF YOURSELF OR AS AN AUTHORIZED REPRESENTATIVE ON BEHALF OF AN ENTITY) AGREE TO ALL THE TERMS OF THIS END USER LICENSE AGREEMENT ("LICENSE") REGARDING YOUR USE OF THE SOFTWARE. IF YOU DO NOT AGREE WITH ALL OF THE TERMS OF THIS LICENSE, DO NOT INSTALL, COPY OR OTHERWISE USE THE SOFTWARE AND IMMEDIATELY CONTACT RABBIT FOR RETURN OF THE SOFTWARE AND A REFUND OF THE PURCHASE PRICE FOR THE SOFTWARE.

We are sorry about the formality of the language below, which our lawyers tell us we need to include to protect our legal rights. If You have any questions, write or call Rabbit at (530) 757-4616, 2900 Spafford Street, Davis, California 95616.

1. **Definitions.** In addition to the definitions stated in the first paragraph of this document, capitalized words used in this License shall have the following meanings:
  - 1.1 "Qualified Applications" means an application program developed using the Software and that links with the development libraries of the Software.
    - 1.1.1 "Qualified Applications" is amended to include application programs developed using the Softools WinIDE program for Rabbit processors available from Softools, Inc.
    - 1.1.2 The MicroC/OS-II ( $\mu$ C/OS-II) library and sample code and the Point-to-Point Protocol (PPP) library are not included in this amendment.
    - 1.1.3 Excluding the exceptions in 1.1.2, library and sample code provided with the Software may be modified for use with the Softools WinIDE program in Qualified Systems as defined in 1.2. All other Restrictions specified by this license agreement remain in force.
  - 1.2 "Qualified Systems" means a microprocessor-based computer system which is either (i) manufactured by, for or under license from Rabbit, or (ii) based on the Rabbit 2000 microprocessor, the Rabbit 3000 microprocessor, the Rabbit 4000 microprocessor, or any other Rabbit microprocessor. Qualified Systems may not be (a) designed or intended to be re-programmable by your customer using the Software, or (b) competitive with Rabbit products, except as otherwise stated in a written agreement between Rabbit and the system manufacturer. Such written agreement may require an end user to pay run time royalties to Rabbit.

2. **License.** Rabbit grants to You a nonexclusive, nontransferable license to (i) use and reproduce the Software, solely for internal purposes and only for the number of users for which You have purchased licenses for (the "Users") and not for redistribution or resale; (ii) use and reproduce the Software solely to develop the Qualified Applications; and (iii) use, reproduce and distribute, the Qualified Applications, in object code only, to end users solely for use on Qualified Systems; provided, however, any agreement entered into between You and such end users with respect to a Qualified Application is no less protective of Rabbit's intellectual property rights than the terms and conditions of this License. (iv) use and distribute with Qualified Applications and Qualified Systems the program files distributed with Dynamic C named RFU.EXE, PILOT.BIN, and COLDLOAD.BIN in their unaltered forms.
3. **Restrictions.** Except as otherwise stated, You may not, nor permit anyone else to, decompile, reverse engineer, disassemble or otherwise attempt to reconstruct or discover the source code of the Software, alter, merge, modify, translate, adapt in any way, prepare any derivative work based upon the Software, rent, lease network, loan, distribute or otherwise transfer the Software or any copy thereof. You shall not make copies of the copyrighted Software and/or documentation without the prior written permission of Rabbit; provided that, You may make one (1) hard copy of such documentation for each User and a reasonable number of back-up copies for Your own archival purposes. You may not use copies of the Software as part of a benchmark or comparison test against other similar products in order to produce results strictly for purposes of comparison. The Software contains copyrighted material, trade secrets and other proprietary material of Rabbit and/or its licensors and You must reproduce, on each copy of the Software, all copyright notices and any other proprietary legends that appear on or in the original copy of the Software. Except for the limited license granted above, Rabbit retains all right, title and interest in and to all intellectual property rights embodied in the Software, including but not limited to, patents, copyrights and trade secrets.
4. **Export Law Assurances.** You agree and certify that neither the Software nor any other technical data received from Rabbit, nor the direct product thereof, will be exported outside the United States or re-exported except as authorized and as permitted by the laws and regulations of the United States and/or the laws and regulations of the jurisdiction, (if other than the United States) in which You rightfully obtained the Software. The Software may not be exported to any of the following countries: Cuba, Iran, Iraq, Libya, North Korea, Sudan, or Syria.
5. **Government End Users.** If You are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees: (i) if the Software is supplied to the Department of Defense ("DOD"), the Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and (ii) if the Software is supplied to any unit or agency of the United States Government other than DOD, the Government's rights in the Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

6. **Disclaimer of Warranty.** You expressly acknowledge and agree that the use of the Software and its documentation is at Your sole risk. THE SOFTWARE, DOCUMENTATION, AND TECHNICAL SUPPORT ARE PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND. Information regarding any third party services included in this package is provided as a convenience only, without any warranty by Rabbit, and will be governed solely by the terms agreed upon between You and the third party providing such services. RABBIT AND ITS LICENSORS EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. RABBIT DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. FURTHERMORE, RABBIT DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY RABBIT OR ITS AUTHORIZED REPRESENTATIVES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.
7. **Limitation of Liability.** YOU AGREE THAT UNDER NO CIRCUMSTANCES, INCLUDING NEGLIGENCE, SHALL RABBIT BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE AND/OR INABILITY TO USE THE SOFTWARE, EVEN IF RABBIT OR ITS AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT SHALL RABBIT'S TOTAL LIABILITY TO YOU FOR ALL DAMAGES, LOSSES, AND CAUSES OF ACTION (WHETHER IN CONTRACT, TORT, INCLUDING NEGLIGENCE, OR OTHERWISE) EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE.
8. **Termination.** This License is effective for the duration of the copyright in the Software unless terminated. You may terminate this License at any time by destroying all copies of the Software and its documentation. This License will terminate immediately without notice from Rabbit if You fail to comply with any provision of this License. Upon termination, You must destroy all copies of the Software and its documentation. Except for Section 2 ("License"), all Sections of this Agreement shall survive any expiration or termination of this License.

9. **General Provisions.** No delay or failure to take action under this License will constitute a waiver unless expressly waived in writing, signed by a duly authorized representative of Rabbit, and no single waiver will constitute a continuing or subsequent waiver. This License may not be assigned, sublicensed or otherwise transferred by You, by operation of law or otherwise, without Rabbit's prior written consent. This License shall be governed by and construed in accordance with the laws of the United States and the State of California, exclusive of the conflicts of laws principles. The United Nations Convention on Contracts for the International Sale of Goods shall not apply to this License. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to affect the intent of the parties, and the remainder of this License shall continue in full force and effect. This License constitutes the entire agreement between the parties with respect to the use of the Software and its documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. There shall be no contract for purchase or sale of the Software except upon the terms and conditions specified herein. Any additional or different terms or conditions proposed by You or contained in any purchase order are hereby rejected and shall be of no force and effect unless expressly agreed to in writing by Rabbit. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Rabbit.

Digi International Inc. © 2008 • All rights reserved.