



**For Rabbit Semiconductor Microprocessors
Integrated C Development System**

Function Reference Manual

019-0113 • 040930 - F

Table of Contents

Alphabetical Listing of Dynamic C Functions	v
Group Listing of Dynamic C Functions	xiii
Arithmetic	
Bit Manipulation	
Character	
Data Encryption	
Dynamic Memory Allocation	
ECC	
Error Handling	
Extended Memory	
Fast Fourier Transforms	
File Compression	
File System	
Floating-Point Math	
Global Positioning System	
HDLC Protocol (Rabbit 3000)	
I/O	
I2C Protocol	
Interrupts	
Low-Level Flash Access	
MD5	
MicroC/OS-II	
Miscellaneous	
Multitasking	
NAND Flash	
Number-to-String Conversion	
Partitions	
Pulse Width Modulation (Rabbit 3000)	
Quadrature Decoder (Rabbit 3000)	
Rabbit 3000	
Real-Time Clock	
Serial Communication	
Serial Flash	
Serial Packet Driver	
SPI	
Stdio	
String Manipulation	
String-to-Number Conversion	
System	
User Block	
Watchdogs	
Chapter 1: Function Descriptions	1
Notice to Users	423
Z-World Software End User License Agreement	425

Alphabetical Listing of Dynamic C Functions

Symbols

_sysIsSoftReset	384
_xalloc	407
_xavail	410

A

abs	1
acos	2
acot	3
acsc	4
AESdecrypt	5
AESdecryptStream	6
AESencrypt	7
AESencryptStream	8
AESexpandKey	9
AESinitStream	10
asec	11
asin	12
atan	13
atan2	14
atof	15
atoi	16
atol	17

B

BIT	19
bit	18
BitRdPortE	20
BitRdPortI	21
BitWrPortE	22
BitWrPortI	23

C

CalculateECC256	24
ceil	26
ChkCorrectECC256	25
chkHardReset	27
chkSoftReset	27
chkWDTO	28
clockDoublerOff	29
clockDoublerOn	29
CloseInputCompressedFile	30
CloseOutputCompressedFile	30
CoBegin	31
cof_pktXreceive	31
cof_pktXsend	32
cof_seEread	37
cof_serAgetc	33

cof_serAgetc	34
cof_serAputc	35
cof_serAputs	36
cof_serAread	37
cof_serAwrite	38
cof_serBgetc	33
cof_serBgets	34
cof_serBputc	35
cof_serBputs	36
cof_serBread	37
cof_serBwrite	38
cof_serCgetc	33
cof_serCgets	34
cof_serCputc	35
cof_serCputs	36
cof_serCread	37
cof_serCwrite	38
cof_serDgetc	33
cof_serDgets	34
cof_serDputc	35
cof_serDputs	36
cof_serDread	37
cof_serDwrite	38
cof_serEgetc	33
cof_serEgets	34
cof_serEputs	36
cof_serEwrite	38
cof_serFgetc	33
cof_serFgets	34
cof_serFputs	36
cof_serFread	37
cof_serFwrite	38
CompressFile	39
CoPause	40
CoReset	40
CoResume	41
cos	42
cosh	42

D

DecompressFile	43
defineErrorHandler	44
deg	45
DelayMs	46
DelaySec	47
DelayTicks	47
Disable_HW_WDT	48

E

errlogFormatEntry	51
errlogFormatRegDump	52
errlogFormatStackDump	53
errlogGetHeaderInfo	49
errlogGetMessage	54
errlogGetNthEntry	50
errlogReadHeader	54
exception	55
exit	56
exp	57

F

fabs	58
fclose	59
fcreate	60
fcreate (FS2)	61
fcreate_unused	62
fcreate_unused (FS2)	63
fdelete	64
fdelete (FS2)	65
fflush (FS2)	66
fftcplx	67
fftcplxinv	68
fftrealm	69
fftrealminv	70
flash_erasechip	71
flash_erasesector	72
flash_gettype	73
flash_init	74
flash_read	75
flash_readsector	76
flash_sector2xwindow	77
flash_writesector	78
floor	79
fmod	80
fopen_rd (FS1)	81
fopen_rd (FS2)	82
fopen_wr	83
fopen_wr (FS2)	84
forceSoftReset	85
fread	86
fread (FS2)	87
frexp	88
fs_format (FS1)	89
fs_format (FS2)	90
fs_get_flash_lx (FS2)	97
fs_get_lx (FS2)	98
fs_get_lx_size (FS2)	99
fs_get_other_lx (FS2)	100
fs_get_ram_lx (FS2)	101
fs_init (FS1)	91
fs_init (FS2)	92

fs_reserve_blocks (FS1)	93
fs_set_lx (FS2)	102
fs_setup (FS2)	103
fs_sync (FS2)	105
fsck	94
fseek (FS1)	95
fseek (FS2)	96
fshift	108
ftell (FS1)	106
ftell (FS2)	107
ftoa	112
fwrite (FS1)	110
fwrite (FS2)	111

G

getchar	113
getcrc	114
getdivider19200	115
gets	116
GetVectExtern2000	117
GetVectExtern3000	118
GetVectIntern	119
gps_get_position	120
gps_get_utc	121
gps_ground_distance	122

H

hanncplx	123
hannreal	124
HDLCDropX	125
HDLCErrrorX	126
HDLCOpenX	127
HDLCPeekX	128
HDLCreceiveX	129
HDLCSendingX	131
HDLCSendX	130
hitwd	131
htoa	132

I

i2c_check_ack	144
i2c_init	145
i2c_read_char	145
i2c_send_ack	146
i2c_send_nak	146
i2c_start_tx	147
i2c_startw_tx	148
i2c_stop_tx	149
i2c_write_char	149
IntervalMs	133
IntervalSec	133
IntervalTick	134
ipres	134

ipset	135
isalnum	135
isalpha	136
iscntrl	136
isCoDone	137
isCoRunning	137
isdigit	138
isgraph	139
islower	139
isprint	140
ispunct	141
isspace	140
isupper	142
isxdigit	142
itoa	143

K

kbhit	150
-------------	-----

L

labs	150
ldexp	151
log	151
log10	152
longjmp	152
loophead	153
loopinit	153
lsqrt	154
ltoa	154
ltoan	155
lx_format	156

M

mbr_CreatePartition	157
mbr_EnumDrive	158
mbr_FormatDrive	159
mbr_MountPartition	160
mbr_UnmountPartition	161
mbr_ValidatePartitions	162
md5_append	163
md5_finish	164
md5_ini	163
memchr	164
memcmp	165
memcpy	166
memmove	167
memset	167
mktime	168
mktm	169
modf	170

N

nf_eraseBlock	171
---------------------	-----

nf_getPageCount	172
nf_getPageSize	173
nf_initDevice	174
nf_InitDriver	176
nf_isBusyRBHW	177
nf_isBusyStatus	178
nf_readPage	179
nf_writePage	180

O

OpenInputCompressedFile	181
OpenOutputCompressedFile ..	182
OS_ENTER_CRITICAL	183
OS_EXIT_CRITICAL	183
OSFlagAccept	184
OSFlagCreate	186
OSFlagDel	187
OSFlagPend	188
OSFlagPost	190
OSFlagQuery	191
OSInit	192
OSMboxAccept	192
OSMboxCreate	193
OSMboxDel	194
OSMboxPend	195
OSMboxPost	196
OSMboxPostOpt	197
OSMboxQuery	198
OSMemCreate	199
OSMemGet	200
OSMemPut	201
OSMemQuery	202
OSMutexAccept	203
OSMutexCreate	204
OSMutexDel	205
OSMutexPend	206
OSMutexPost	207
OSMutexQuery	208
OSQAccept	209
OSQCreate	210
OSQDel	211
OSQFlush	212
OSQPend	213
OSQPost	214
OSQPostFront	215
OSQPostOpt	216
OSQQuery	217
OSSchedLock	218
OSSchedUnlock	218
OSSemAccept	219
OSSemCreate	220
OSSemPend	221
OSSemPost	222
OSSemQuery	223

OSSetTickPerSec	224
OSStart	224
OSStatInit	225
OSTaskChangePrio	225
OSTaskCreate	226
OSTaskCreateExt	227
OSTaskCreateHook	228
OSTaskDel	229
OSTaskDelHook	230
OSTaskDelReq	231
OSTaskIdleHook	232
OSTaskQuery	232
OSTaskResume	233
OSTaskStatHook	233
OSTaskStkChk	234
OSTaskSuspend	235
OSTaskSwHook	236
OSTCBInitHook	236
OSTimeDly	237
OSTimeDlyHMSM	238
OSTimeDlyResume	239
OSTimeDlySec	240
OSTimeGet	241
OSTimeSet	241
OSTimeTick	242
OSTimeTickHook	242
OSVersion	243
outchrs	243
outstr	244

P

paddr	245
paddrDS	246
paddrSS	247
palloc	248
palloc_fast	249
pavail	250
pavail_fast	251
pccalloc	252
pfirst	253
pfirst_fast	254
pfree	255
pfree_fast	256
phwm	257
pktXclose	258
pktXgetErrors	258
pktXinitBuffers	259
pktXopen	260
pktXreceive	262
pktXsend	263
pktXsending	264
pktXsetParity	264
plast	265
plast_fast	266

pmovebetween	267
pmovebetween_fast	269
pncl	270
pnext	271
pnext_fast	272
poly	273
pool_append	274
pool_init	275
pool_link	276
pool_xappend	277
pool_xinit	278
pow	279
pow10	279
powerspectrum	280
pprev	281
pprev_fast	282
premain	283
preorder	284
printf	286
putchar	292
puts	293
pwm_init	293
pwm_set	294
pxalloc	295
pxalloc_fast	296
pxcalloc	297
pxfirst	298
pxfirst_fast	299
pxfree	300
pxfree_fast	301
pxlast	302
pxlast_fast	303
pxnext	304
pxnext_fast	305
pxprev	306
pxprev_fast	307

Q

qd_error	308
qd_init	309
qd_read	309
qd_zero	310
qsort	311

R

rad	312
rand	313
randb	313
randg	314
RdPortE	314
RdPortI	315
read_rtc	317
read_rtc_32kHz	317
ReadCompressedFile	316

readUserBlock	318
readUserBlockArray	319
RES	321
res	320
ResetErrorLog	322
root2xmem	322
rtc_timezone	323
runwatch	324

S

serAclose	325
serAdatabits	325
serAflowcontrolOff	326
serAflowcontrolOn	327
serAgetc	328
serAgetError	329
serAopen	330
serAparity	331
serApeek	332
serAputc	333
serAputs	334
serArdFlush	335
serArdFree	335
serArdUsed	336
serAread	337
serAwrFlush	338
serAwrFree	338
serAwrite	339
serBclose	325
serBdatabits	325
serBflowcontrolOff	326
serBflowcontrolOn	327
serBgetc	328
serBgetError	329
serBopen	330
serBparity	331
serBpeek	332
serBputc	333
serBputs	334
serBrdFlush	335
serBrdFree	335
serBrdUsed	336
serBread	337
serBwrFlush	338
serBwrFree	338
serBwrite	339
serCclose	325
serCdatabits	325
serCflowcontrolOff	326
serCflowcontrolOn	327
serCgetc	328
serCgetError	329
serCheckParity	324
serCopen	330
serCparity	331
serCpeek	332
serCputc	333
serCputs	334
serCrdFlush	335
serCrdFree	335
serCrdUsed	336
serCread	337
serCwrFlush	338
serCwrFree	338
serCwrite	339
serDclose	325
serDdatabits	325
serDflowcontrolOff	326
serDflowcontrolOn	327
serDgetc	328
serDgetError	329
serDopen	330
serDparity	331
serDpeek	332
serDputc	333
serDputs	334
serDrdFlush	335
serDrdFree	335
serDrdUsed	336
serDread	337
serDwrFlush	338
serDwrFree	338
serDwrite	339
serEclose	325
serEdatabits	325
serEflowcontrolOff	326
serEflowcontrolOn	327
serEgetc	328
serEgetError	329
serEopen	330
serEparity	331
serEpeek	332
serEputc	333
serEputs	334
serErdFlush	335
serErdFree	335
serErdUsed	336
serEread	337
serEwrFlush	338
serEwrFree	338
serEwrite	339
serFclose	325
serFdatabits	325
serFflowcontrolOff	326
serFflowcontrolOn	327
serFgetc	328
serFgetError	329
serFopen	330

serFparity	331
serFpeek	332
serFputc	333
serFputs	334
serFrdFlush	335
serFrdFree	335
serFrdUsed	336
serFread	337
serFwrFlush	338
serFwrFree	338
serFwrite	339
SET	341
set	340
set32kHzDivider	342
setClockModulation	343
setjmp	344
SetVectExtern2000	345
SetVectExtern3000	346
SetVectIntern	347
sf_getPageCount	349
sf_getPageSize	349
sf_init	350
sf_initDevice	351
sf_isWriting	352
sf_pageToRAM	352
sf_RAMToPage	353
sf_readDeviceRAM	354
sf_readPage	355
sf_readRAM	356
sf_writeDeviceRAM	357
sf_writePage	358
sf_writeRAM	359
sfspi_init	360
sin	360
sinh	361
snprintf	362
SPIinit	363
SPIRead	364
SPIWrite	365
SPIWrRd	366
sprintf	367
sqrt	368
srand	368
strcat	369
strchr	370
strcmp	371
strcmpi	372
strcpy	373
strcspn	373
strlen	374
strncat	374
strncmp	375
strncmpi	376
strncpy	377

strpbrk	378
strchr	379
strspn	379
strstr	380
strtod	381
strtok	382
strtol	383
sysResetChain	384

T

tan	385
tanh	386
tm_rd	387
tm_wr	388
tolower	389
toupper	389

U

updateTimers	390
use32kHzOsc	391
useClockDivider	392
useClockDivider3000	393
useMainOsc	394
utoa	394

V

VdGetFreeWd	395
VdInit	396
VdReleaseWd	397

W

write_rtc	400
WriteFlash2	398
WriteFlash2Array	399
writeUserBlock	401
writeUserBlockArray	403
WrPortE	404
WrPortI	405

X

xalloc	406
xalloc_stats	408
xavail	409
xCalculateECC256	411
xChkCorrectECC256	412
xgetfloat	413
xgetint	413
xgetlong	414
xmem2root	415
xmem2xmem	416
xmemchr	417
xmemcmp	418
xrelease	419

xsetfloat	421
xsetint	420
xsetlong	421
xstrlen	422

Group Listing of Dynamic C Functions

A

Arithmetic

abs	1
geterc	114
lsqrt	154

B

Bit Manipulation

BIT	19
bit	18
RES	321
res	320
SET	341
set	340

C

Character

isalnum	135
isalpha	136
isctrl	136
isdigit	138
isgraph	139
islower	139
isprint	140
ispunct	141
isspace	140
isupper	142
isxdigit	142

D

Data Encryption

AESdecrypt	5
AESdecryptStream	6
AESencrypt	7

AESencryptStream	8
AESexpandKey	9
AESinitStream	10

Dynamic Memory Allocation

palloc	248
palloc_fast	249
pavail	250
pavail_fast	251
pcalloc	252
pfirst	253
pfirst_fast	254
pfree	255
pfree_fast	256
phwm	257
plast	265
plast_fast	266
pmovebetween	267
pmovebetween_fast	269
pnel	270
pnext	271
pnext_fast	272
pool_append	274
pool_init	275
pool_link	276
pool_xappend	277
pool_xinit	278
pprev	281
pprev_fast	282
preorder	284
pxalloc	295
pxalloc_fast	296
pxcalloc	297
pxfirst	298
pxfirst_fast	299
pxfree	300

pxfree_fast	301
pxlast	302
pxlast_fast	303
pxnext	304
pxnext_fast	305
pxprev	306
pxprev_fast	307

E

ECC

CalculateECC256	24
ChkCorrectECC256	25
xCalculateECC256	411
xChkCorrectECC256	412

Error Handling

errlogFormatEntry	51
errlogFormatRegDump	52
errlogFormatStackDump	53
errlogGetHeaderInfo	49
errlogGetMessage	54
errlogGetNthEntry	50
errlogReadHeader	54
exception	55
ResetErrorLog	322

Extended Memory

_xalloc	407
_xavail	410
paddr	245
paddrDS	246
paddrSS	247
root2xmem	322
WriteFlash2	398
WriteFlash2Array	399
xalloc	406
xalloc_stats	408
xavail	409
xgetfloat	413

xgetint	413
xgetlong	414
xmem2root	415
xmem2xmem	416
xmemchr	417
xmemcmp	418
xrelease	419
xsetfloat	421
xsetint	420
xsetlong	421
xstrlen	422

F

Fast Fourier Transforms

fftcplx	67
fftcplxinv	68
fftrealm	69
fftrealminv	70
hanncplx	123
hannreal	124
powerspectrum	280

File Compression

CloseInputCompressedFile	30
CloseOutputCompressedFile	30
CompressFile	39
DecompressFile	43
OpenInputCompressedFile	181
OpenOutputCompressedFile	182
ReadCompressedFile	316

File System

fclose	59
fcreate (FS1)	60
fcreate (FS2)	61
fcreate_unused (FS1)	62
fcreate_unused (FS2)	63
fdelete (FS1)	64
fdelete (FS2)	65

fflush (FS2)	66	cos	42
fopen_rd (FS1)	81	cosh	42
fopen_rd (FS2)	82	deg	45
fopen_wr (FS1)	83	exp	57
fopen_wr (FS2)	84	fabs	58
fread (FS1)	86	floor	79
fread (FS2)	87	fmod	80
fs_format (FS1)	89	frexp	88
fs_format (FS2)	90	labs	150
fs_get_flash_lx (FS2)	97	ldexp	151
fs_get_lx (FS2)	98	log	151
fs_get_lx_size (FS2)	99	log10	152
fs_get_other_lx (FS2)	100	modf	170
fs_get_ram_lx (FS2)	101	poly	273
fs_init (FS1)	91	pow	279
fs_init (FS2)	92	pow10	279
fs_reserve_blocks (FS1)	93	rad	312
fs_set_lx (FS2)	102	rand	313
fs_setup (FS2)	103	randb	313
fs_sync (FS2)	105	randg	314
fsck (FS1)	94	sin	360
fseek (FS1)	95	sinh	361
fseek (FS2)	96	sqrt	368
fshift	108	srand	368
ftell (FS1)	106	tan	385
ftell (FS2)	107	tanh	386
fwrite (FS1)	110		
fwrite (FS2)	111		
lx_format	156		

Floating-Point Math

acos	2
acot	3
acsc	4
asec	11
asin	12
atan	13
atan2	14
ceil	26

G

Global Positioning System

gps_get_position	120
gps_get_utc	121
gps_ground_distance	122

H

HDLC Protocol (Rabbit 3000)

HDLCdropX	125
HDLCerrorX	126
HDLCopenX	127

HDLCpeekX	128
HDLCreceiveX	129
HDLCsendingX	131
HDLCsendX	130

I

I/O

BitRdPortE	20
BitRdPortI	21
BitWrPortE	22
BitWrPortI	23
RdPortE	314
RdPortI	315
WrPortE	404
WrPortI	405

I2C Protocol

i2c_check_ack	144
i2c_init	145
i2c_read_char	145
i2c_send_ack	146
i2c_send_nak	146
i2c_start_tx	147
i2c_startw_tx	148
i2c_stop_tx	149
i2c_write_char	149

Interrupts

GetVectExtern300	118
ipres	134
ipset	135
SetVectExtern2000	345
SetVectExtern3000	346
SetVectIntern	347

L

Low-Level Flash Access

flash_erasechip	71
flash_erasesector	72

flash_gettype	73
flash_init	74
flash_read	75
flash_readsector	76
flash_sector2xwindow	77
flash_writesector	78

M

MD5

md5_append	163
md5_finish	164
md5_init	163

MicroC/OS-II

OOSQDel	211
OS_ENTER_CRITICAL	183
OS_EXIT_CRITICAL	183
OSFlagAccept	184
OSFlagCreate	186
OSFlagDel	187
OSFlagPend	188
OSFlagPost	190
OSFlagQuery	191
OSInit	192
OSMboxAccept	192
OSMboxCreate	193
OSMboxDel	194
OSMboxPend	195
OSMboxPost	196
OSMboxPostOpt	197
OSMboxQuery	198
OSMemCreate	199
OSMemGet	200
OSMemPut	201
OSMemQuery	202
OSMutexAccept	203
OSMutexCreate	204
OSMutexDel	205
OSMutexPend	206

OSMutexPost	207	OSTimeGet	241
OSMutexQuery	208	OSTimeSet	241
OSQAccept	209	OSTimeTick	242
OSQCreate	210	OSTimeTickHook	242
OSQFlush	212	OSVersion	243
OSQPend	213		
OSQPost	214	Miscellaneous	
OSQPostFront	215	longjmp	152
OSQPostOpt	216	qsort	311
OSQQuery	217	runwatch	324
OSSchedLock	218	setjmp	344
OSSchedUnlock	218		
OSSemAccept	219	Multitasking	
OSSemCreate	220	CoBegin	31
OSSemPend	221	CoPause	40
OSSemPost	222	CoReset	40
OSSemQuery	223	CoResume	41
OSSetTickPerSec	224	DelayMs	46
OSStart	224	DelaySec	47
OSStatInit	225	DelayTicks	47
OSTaskChangePrio	225	IntervalMs	133
OSTaskCreate	226	IntervalSec	133
OSTaskCreateExt	227	IntervalTick	134
OSTaskCreateHook	228	isCoDone	137
OSTaskDel	229	isCoRunning	137
OSTaskDelHook	230	loophead	153
OSTaskDelReq	231	loopinit	153
OSTaskIdleHook	232		
OSTaskQuery	232	N	
OSTaskResume	233	NAND Flash	
OSTaskStatHook	233	nf_eraseBlock	171
OSTaskStkChk	234	nf_getPageCount	172
OSTaskSuspend	235	nf_getPageSize	173
OSTaskSwHook	236	nf_initDevice	174
OSTCBInitHook	236	nf_InitDriver	176
OSTimeDly	237	nf_isBusyRBHW	177
OSTimeDlyHMSM	238	nf_isBusyStatus	178
OSTimeDlyResume	239	nf_readPage	179
OSTimeDlySec	240	nf_writePage	180

Number-to-String Conversion

ftoa	112
htoa	132
itoa	143
ltoa	154
ltoan	155
utoa	394

P

Partitions

mbr_CreatePartition	157
mbr_EnumDrive	158
mbr_FormatDrive	159
mbr_MountPartition	160
mbr_UnmountPartition	161
mbr_ValidatePartitions	162

Pulse Width Modulation (Rabbit 3000)

pwm_init	293
pwm_set	294

Q

Quadrature Decoder (Rabbit 3000)

qd_error	308
qd_init	309
qd_read	309
qd_zero	310

R

Rabbit 3000

HDLCDropX	125
HDLCerrorX	126
HDLCopenX	127
HDLCpeekX	128
HDLCreceiveX	129
HDLCsendingX	131
HDLCsendX	130

pwm_init	293
pwm_set	294
qd_error	308
qd_init	309
qd_read	309
qd_zero	310

Real-Time Clock

mktime	168
mktm	169
read_rtc	317
read_rtc_32kHz	317
rtc_timezone	323
tm_rd	387
tm_wr	388
write_rtc	400

S

Serial Communication

cof_serXgetc	33
cof_serXgets	34
cof_serXputc	35
cof_serXputs	36
cof_serXread	37
cof_serXwrite	38
serCheckParity	324
serXclose	325
serXdatabits	325
serXflowcontrolOff	326
serXflowcontrolOn	327
serXgetc	328
serXgetError	329
serXopen	330
serXparity	331
serXpeek	332
serXputc	333
serXputs	334
serXrdFlush	335
serXrdFree	335

serXrdUsed	336	Stdio	
serXread	337	getchar	113
serXwrFlush	338	gets	116
serXwrFree	338	kbhit	150
serXwrite	339	outchrs	243
Serial Flash		outstr	244
sf_getPageCount	349	printf	286
sf_getPageSize	349	putchar	292
sf_init	350	puts	293
sf_initDevice	351	snprintf	362
sf_isWriting	352	sprintf	367
sf_pageToRAM	352	String Manipulation	
sf_RAMToPage	353	memchr	164
sf_readDeviceRAM	354	memcmp	165
sf_readPage	355	memcpy	166
sf_readRAM	356	memmove	167
sf_writeDeviceRAM	357	memset	167
sf_writePage	358	strcat	369
sf_writeRAM	359	strchr	370
sfspi_init	360	strcmp	371
Serial Packet Driver		strcmpi	372
cof_pktXreceive	31	strcpy	373
cof_pktXsend	32	strcspn	373
pktXclose	258	strlen	374
pktXgetErrors	258	strncat	374
pktXinitBuffers	259	strncmp	375
pktXopen	260	strncmpi	376
pktXreceive	262	strncpy	377
pktXsend	263	strpbrk	378
pktXsending	264	strrchr	379
pktXsetParity	264	strspn	379
SPI		strstr	380
SPIinit	363	tolower	389
SPIRead	364	toupper	389
SPIWrite	365	String-to-Number Conversion	
SPIWrRd	366	atof	15
		atoi	16

atol	17
strtod	381
strtok	382
strtol	383

System

_sysIsSoftReset	384
chkHardReset	27
chkSoftReset	27
chkWDTO	28
clockDoublerOff	29
clockDoublerOn	29
defineErrorHandler	44
exit	56
forceSoftReset	85
getdivider19200	115
GetVectExtern2000	117
GetVectIntern	119
ipres	134
ipset	135
premain	283
set32kHzDivider	342
setClockModulation	343
sysResetChain	384
updateTimers	390
use32kHzOsc	391
useClockDivider	392
useClockDivider3000	393
useMainOsc	394

U

User Block

readUserBlock	318
readUserBlockArray	319
writeUserBlock	401
writeUserBlockArray	403

W

Watchdogs

Disable_HW_WDT	48
Enable_HW_WDT	48
hitwd	131
VdGetFreeWd	395
VdHitWd	396
VdInit	396
VdReleaseWd	397

1. Function Descriptions

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

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

state	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>
data	An array of bytes that will be decrypted in place.
count	Size of data array

LIBRARY

`AES_CRYPT.LIB`

AESEncrypt

```
void AESEncrypt( 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

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

RETURN VALUE

None.

LIBRARY

`AES_CRYPT.LIB`

AESencryptStream

```
void AESencryptStream( 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

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

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

state	An AESstreamState structure to be initialized. This memory must be allocated in the program code before calling AESInitStream().
key	The 16-byte cipher key, using a null pointer, will prevent an existing key from being recalculated.
init_vector	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

```
float atof( char *sptr );
```

DESCRIPTION

ANSI String to Float Conversion (UNIX compatible).

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

```
int atoi( char *sptr );
```

DESCRIPTION

ANSI String to Integer Conversion (UNIX compatible).

PARAMETERS

sptr String to convert.

RETURN VALUE

The converted integer value.

LIBRARY

STRING.LIB

SEE ALSO

atol, atof, strtod

atol

```
long atol( char *sptr );
```

DESCRIPTION

ANSI String to Long Conversion (UNIX compatible).

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

address	Address of byte containing bits 7-0
bit	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

address	Address of byte containing bits 7-0
bit	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

port	Address of external parallel port data register.
bitnumber	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

port	Address of internal parallel port data register.
bitnumber	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

port	Address of external parallel port data register.
portshadow	Reference pointer to a variable to shadow the current value of the register.
value	Value of 0 or 1 to be written to the bit position.
bitcode	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

port	Address of internal parallel port data register.
portshadow	Reference pointer to a variable to shadow the current value of the register.
value	Value of 0 or 1 to be written to the bit position.
bitcode	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

data	Pointer to the 256 byte data buffer
old_ecc	Pointer to the old (original) 3 byte ECC's buffer
new_ecc	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 watch-dog 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();
```

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();
```

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=A|B|C|D
```

DESCRIPTION

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

Starting with Dynamic C version 7.25, the functions `cof_pktEreceive()` and `cof_pktFreceive()` are available when using the Rabbit 3000 microprocessor.

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=A|B|C|D
```

DESCRIPTION

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

Starting with Dynamic C version 7.25, the functions `cof_pktEsend()` and `cof_pktFsend()` are available when using the Rabbit 3000 microprocessor.

PARAMETERS

send_buffer	The data to be sent.
buffer_length	Length of the data buffer to transmit.
delay	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(); /* where X = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `cof_serEgetc()` and `cof_serFgetc()` may be used with the Rabbit 3000 microprocessor.

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 = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions cof_serEgets() and cof_serFgets() may be used with the Rabbit 3000 microprocessor.

PARAMETERS

s	Character array into which a null terminated string is read.
max	The maximum number of characters to read into s.
tmout	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 = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `cof_serEputc()` and `cof_serFputc()` may be used with the Rabbit 3000 microprocessor.

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 = A|B|C|D */
```

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.

Starting with Dynamic C version 7.25, the functions `cof_serEputs()` and `cof_serFputs()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

str Null terminated character string to write.

LIBRARY

RS232.LIB

EXAMPLE

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

cof_serXread

```
int cof_serXread( void *data, int length, unsigned long tmout);  
/* where X = A|B|C|D|E|F */
```

DESCRIPTION

This single-user cofunction reads `length` characters from port `X` 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.

Starting with Dynamic C version 7.25, the functions `cof_serEread()` and `cof_serFread()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure into which characters are read.
length	The number of characters to read into <code>data</code> .
tmout	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 = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `cof_serEwrite()` and `cof_serFwrite()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure to write.
length	Number of bytes in data to write.

LIBRARY

RS232.LIB

EXAMPLE

```
// writes a block of characters, repeatedly  
main() {  
    const char s[] = "Hello Z-World";  
    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

input	Input bit file
output	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

input	Input bit file
output	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();
```

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`

Enable_HW_WDT

```
void Enable_HW_WDT();
```

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`

errlogGetHeaderInfo

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

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 WD-TOs 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 WD-TOs 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 <code>errLogEntry</code>
----------	--

RETURN VALUE

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

LIBRARY

`ERRORS.LIB`

errlogFormatEntry

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

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();
```

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();
```

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();
```

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();
```

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`

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`

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

f	The pointer to the created file.
fnum	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

f	Pointer to the file descriptor to fill in.
name	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

fnum	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

name	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

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

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array <code>x</code> .
blockexp	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 f_{max} 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

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

LIBRARY

FFT.LIB

SEE ALSO

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

fftrealignv

```
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

x	Pointer to N -element array of complex fractions.
N	Number of complex elements in array x .
blockexp	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

fd	Pointer to flash descriptor of the chip to erase a sector of.
which	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_erasesector`, `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

fd	This is a pointer to an internal data structure that holds information about a flash memory chip.
mb3cr	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

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

fd	The <code>FlashDescriptor</code> of the flash memory to read from.
sector	The source sector to read.
buffer	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

fd	The FlashDescriptor of the flash memory.
sector	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

fd	The <code>FlashDescriptor</code> of the flash memory to write to.
sector	The destination sector.
buffer	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

f	A pointer to the file to read.
fnum	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

f	Pointer to file descriptor (uninitialized).
name	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

f	A pointer to the file to write.
fnum	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

f	Pointer to file descriptor (uninitialized).
name	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();
```

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

f	A pointer to the file to read from.
buf	A pointer to the destination buffer.
len	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

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

x	Number to split
n	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

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 to allocate for the file system. With a default block size of 4096 bytes and a 256K flash memory, this value might be 64.
wearlevel	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 wearlevel 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

reserveblocks	Must be zero. Retained for backward compatibility.
num_blocks	Ignored (backward compatibility).
wearlevel	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 μ 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

f	A pointer to the file to seek into.
to	The number of bytes to move the read pointer. This can be a positive or negative number.
whence	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

f	Pointer to file descriptor (initialized by <code>fopen_rd()</code> , <code>fopen_wr()</code> or <code>fcreate()</code>).
where	New position, or offset.
whence	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 SEEK_END, 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

meta	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

lxn	Logical extent number to query.
all	Boolean: 0 for current free capacity only, 1 for total. Must use 1 if calling before <code>fs_init()</code> .
ls_shift	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\FS2INFO.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

meta	Extent number for metadata.
data	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

lxn	Base extent number to partition or modify.
ls_shift	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.
reserve_it	TRUE if base partition is to be marked reserved.
rfu	A pointer reserved for future use. Pass as null.
partition_it	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> .

part	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.
part_ls_shift	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.
part_reserve	TRUE if the new extent is to be reserved.
part_rfu	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 `s` is valid. If the File is not actually open, the return value will be random.

PARAMETERS

f	Pointer to file descriptor (initialized by <code>fopen_rd()</code> , <code>fopen_wr()</code> or <code>fcreate()</code>).
----------	---

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

f	Pointer to file descriptor (initialized by <code>fopen_wr()</code> or <code>fcreate()</code>).
len	Length of data to remove (0 to 32767 inclusive).
*buf	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.

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

f	A pointer to the file to write to.
buf	A pointer to the source buffer.
len	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

f	Pointer to file descriptor (initialized by <code>fopen_wr()</code> or <code>fcreate()</code>).
buf	Data buffer located in root data memory or stack.
len	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

dataarray	Data buffer
count	Number of bytes. Maximum is 255.
accum	Base CRC for the data array.

RETURN VALUE

CRC value.

LIBRARY

MATH.LIB

getdivider19200

```
char getdivider19200();
```

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

s	The input string is put to the location pointed to by the argument s. The caller is responsible to make sure the location pointed to by s 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

GetVectExtern2000

```
unsigned GetVectExtern2000();
```

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 2 external interrupts. (Please see Technical Note 301, “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()` really just returns whatever value is at the 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:

$$(\text{internal vector table base}) + (\text{vectNum} * 16) + 1$$

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( GPSPositon *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

newtime	tm structure to fill with new UTC time.
sentence	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

a	First point.
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

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array x.
blockexp	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 *fmax* term. The dc and *fmax* spectral components take part in the convolution along with the other spectral components. The real part of *fmax* 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

x	Pointer to N-element array of complex fractions.
N	Number of complex elements in array <code>x</code> .
blockexp	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`

HDLCdropX

```
int HDLCdropX(); /* 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 microprocessor.

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 microprocessor.

RETURN VALUE

Error flags (see above).

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 microprocessor. Please see the *Rabbit 3000 Microprocessor User's Manual* for more details on HDLC and the bit encoding modes to use.

PARAMETERS

baud	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.
encoding	The bit encoding mode to use. Macro labels for the available options are: <ul style="list-style-type: none">• HDLC_NRZ• HDLC_NRZI• HDLC_MANCHESTER• HDLC_BIPHASE_SPACE• HDLC_BIPHASE_MARK
buffers	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: $(\text{\# of buffers}) * ((\text{size of buffer}) + 4)$
buffer_count	The number of buffers in the block pointed to by <code>buffer</code> .
buffer_size	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

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 microprocessor.

PARAMETERS

bufptr	Pointer to location in xmem of the received packet.
lenptr	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

`HDLCPACKET.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 microprocessor.

PARAMETERS

rx_buffer	Pointer to the buffer to copy a received packet into.
length	Size of the buffer pointed to by <code>rx_buffer</code> .

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

`HDL_C_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 HDLCsendingX() returns false, indicating that the driver is done with it.

This function is intended for use with the Rabbit 3000 microprocessor.

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(); /* 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 microprocessor.

RETURN VALUE

1: Currently sending a packet.
0: Transmitter is idle.

LIBRARY

HDLC_PACKET.LIB

hitwd

```
void hitwd();
```

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

hton

```
char *hton( int value, char *buf );
```

DESCRIPTION

Converts integer value to hexadecimal number and puts result into buf.

PARAMETERS

value	16-bit number to convert
buf	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

value	16-bit signed number to convert
buf	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();
```

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();
```

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 I²C, but allows I²C 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();
```

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();
```

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();
```

DESCRIPTION

Initiates I²C 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();
```

DESCRIPTION

Initiates I²C 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();
```

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();
```

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

x	The value between 0.5 inclusive, and 1.0
n	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

x	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

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 $-\text{INF}$ 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.

PARAMETERS

env	Environment previously saved with <code>setjmp()</code> .
val	Integer result of <code>setjmp()</code> .

LIBRARY

SYS.LIB

SEE ALSO

setjmp

loophead

```
void loophead();
```

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();
```

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

lxn	Logical extent number (1.. <code>fs.num_lx</code> inclusive).
wearlevel	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 = PARTTYPE_NONFSDATA;
drive.part[partnum].endhead = 0xfe;
drive.part[partnum].endseccyl = 0;
drive.part[partnum].startsector = start;
drive.part[partnum].partseclsize = ((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

PARAMETERS

drive	Pointer to a MBR drive structure
pnum	Partition number to be created or modified
type	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.

LIBRARY

PART.LIB

mbr_EnumDrive

```
int mbr_EnumDrive( dos_ctrl *ctrl, mbr_drive *drive, int
    drvnum, int (*checktype)() );
```

DESCRIPTION

This routine is called to learn about drives present on the controller passed in. The drive will be added to the linked list of enumerated drives. Partition information will be filled in from the master boot record. Pointers to file system level partition information structures will be set to null.

PARAMETERS

ctrl	Pointer to a DOS controller structure (set up during initialization of storage device driver.)
drive	Pointer to a drive structure to be filled in
drvnum	Physical drive number of drive on the controller.
checktype	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 drive/device or structure.
-EINVAL if drvnum invalid or does not exist.
-ENOMEM if memory for page buffer is not available.
-EUNFORMAT if the drive 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 drive is invalid
-ENOPART if the drive does not have any sought partitions, If checktype parameter is null, this test is bypassed. This code is superceded by any other error detected.
-EPERM if the drive has already been enumerated.
-EBUSY if the device is busy.

LIBRARY

PART.LIB

SEE ALSO

mbr_FormatDrive

```
int mbr_FormatDrive( mbr_drive *drive );
```

DESCRIPTION

Creates or rewrites the Master Boot Record (MBR) on the drive given. The routine will only rewrite the Boot Loader code if an MBR already exists on the drive. 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.

PARAMETERS

drive	Pointer to a drive structure
--------------	------------------------------

RETURN VALUE

0 for success

- EEXIST if the MBR exists, writing Boot Loader only
- EIO for Error trying to read the drive/device or structure.
- EINVAL if the Drive structure is not valid.
- ENOMEM if memory for page buffer is not available.
- EPERM if drive has mounted or FS enumerated partition(s)
- EBUSY if the device is busy.

LIBRARY

PART.LIB

SEE ALSO

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

drive	Pointer to a drive structure
pnum	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`

SEE ALSO

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

drive	Pointer to a drive structure containing the partition
pnum	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`

SEE ALSO

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

SEE ALSO

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

md5_append	Pointer to the md5_state_t structure that was initialized by md5_init.
data	Pointer to the data to be hashed.
nbytes	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

pms	Pointer to the md5_state_t 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

pms	Pointer to the <code>md5_state_t</code> structure that was initialized by <code>md5_init</code> .
digest	The 16-byte array that the hash value will be written into.

LIBRARY

MD5.LIB

memchr

```
void *memchr( void *src, int ch, unsigned int n );
```

DESCRIPTION

Searches up to `n` characters at memory pointed to by `src` for character `ch`.

PARAMETERS

src	Pointer to memory source.
ch	Character to search for.
n	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

```
int memcmp( void *s1, void *s2, size_t n );
```

DESCRIPTION

Performs unsigned character by character comparison of two memory blocks of length `n`.

PARAMETERS

s1	Pointer to block 1.
s2	Pointer to block 2.
n	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

```
void *memcpy( void *dst, void *src, unsigned int n );
```

DESCRIPTION

Copies a block of bytes from one destination to another. Overlap is handled correctly.

PARAMETERS

dst	Pointer to memory destination
src	Pointer to memory source
n	Number of characters to copy

RETURN VALUE

Pointer to destination.

LIBRARY

STRING.LIB

SEE ALSO

memmove, memset

memmove

```
void *memmove( void *dst, void *src, unsigned int n );
```

DESCRIPTION

Copies a block of bytes from one destination to another. Overlap is handled correctly.

PARAMETERS

dst	Pointer to memory destination
src	Pointer to memory source
n	Number of characters to copy

RETURN VALUE

Pointer to destination.

LIBRARY

STRING.LIB

SEE ALSO

memcpy, memset

memset

```
void *memset( void *dst, int chr, unsigned int n );
```

DESCRIPTION

Sets the first *n* bytes of a block of memory pointed to by *dst* to the character *chr*.

PARAMETERS

dst	Block of memory to set
chr	Character that will be written to memory
n	Amount of bytes to set

RETURN VALUE

dst: Pointer to block of memory.

LIBRARY

STRING.LIB

mktime

```
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 <code>tm</code> structure
----------------	--------------------------------------

RETURN VALUE

Time in seconds since January 1, 1980.

LIBRARY

RTCLock.LIB

SEE ALSO

`mktime`, `tm_rd`, `tm_wr`

mktime

```
unsigned int mktime( 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

timeptr	Address to store date and time into structure:
time	Seconds since January 1, 1980.

RETURN VALUE

0

LIBRARY

RTCLOCK.LIB

SEE ALSO

mktime, tm_rd, tm_wr

modf

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

DESCRIPTION

Splits x into a fraction and integer, $f + n$.

PARAMETERS

x	Floating-point integer
n	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

dev	Pointer to an initialized <code>nf_device</code> structure
page	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 <code>nf_device</code> 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 <code>nf_device</code> 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 (continued)

PARAMETERS

dev	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.
which	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

driver	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 Z-World website.
device_list	<p>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>.</p> <p>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.</p>

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

dev	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

dev	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

dev	Pointer to an initialized <code>nf_device</code> structure
dev	Physical address of the xmem buffer to read data into
page	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

dev	Pointer to an initialized <code>nf_device</code> structure
dev	Physical address of the xmem data to be written
page	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`

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

ifp	ZFILE file descriptor
fn	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

ofp	ZFILE file descriptor
fn	FS2 handle of output file

RETURN VALUE

0: Failure
1: Success

LIBRARY

`LZSS.LIB`

SEE ALSO

`CloseOutputCompressedFile`

OS_ENTER_CRITICAL

```
void OS_ENTER_CRITICAL();
```

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();
```

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

pgrp	Pointer to the desired event flag group.
flags	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 <code>flags</code> should be 0x03.
wait_type	<p>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:</p> <ul style="list-style-type: none">• <code>OS_FLAG_WAIT_CLR_ALL</code> - You will check ALL bits in <code>flags</code> to be clear (0)• <code>OS_FLAG_WAIT_CLR_ANY</code> - You will check ANY bit in <code>flags</code> to be clear (0)• <code>OS_FLAG_WAIT_SET_ALL</code> - You will check ALL bits in <code>flags</code> to be set (1)• <code>OS_FLAG_WAIT_SET_ANY</code> - You will check ANY bit in <code>flags</code> to be set (1)

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 (continued)

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

flags	Contains the initial value to store in the event flag group.
err	Pointer to an error code that will be returned to your application: <ul style="list-style-type: none">• OS_NO_ERR - The call was successful.• OS_ERR_CREATE_ISR - Attempt made to create an Event Flag from an ISR.• OS_FLAG_GRP_DEPLETED - There are no more event flag groups

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

pgrp	Pointer to the desired event flag group.
opt	May be one of the following delete options: <ul style="list-style-type: none">• OS_DEL_NO_PEND - Deletes the event flag group only if no task pending• OS_DEL_ALWAYS - Deletes the event flag group even if tasks are waiting. In this case, all the tasks pending will be readied..
err	Pointer to an error code. May be one of the following values: <ul style="list-style-type: none">• OS_NO_ERR - Success, the event flag group was deleted• OS_ERR_DEL_ISR - If you attempted to delete the event flag group 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_ERR_EVENT_TYPE - If you didn't pass a pointer to an event flag group• OS_ERR_INVALID_OPT - Invalid option was specified• OS_ERR_TASK_WAITING - One or more tasks were waiting on the event flag group.

RETURN VALUE

pevent	Error.
(OS_EVENT *) 0	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

pgrp	Pointer to the desired event flag group.
flags	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.
wait_type	<p>Specifies whether you want ALL bits to be set or ANY of the bits to be set. You can specify the following argument:</p> <ul style="list-style-type: none">• <code>OS_FLAG_WAIT_CLR_ALL</code> - You will wait for ALL bits in mask to be clear (0)• <code>OS_FLAG_WAIT_SET_ALL</code> - You will wait for ALL bits in mask to be set (1)• <code>OS_FLAG_WAIT_CLR_ANY</code> - You will wait for ANY bit in mask to be clear (0)• <code>OS_FLAG_WAIT_SET_ANY</code> - You will wait for ANY bit in mask to be set (1) <p>Note: 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> <p style="text-align: center;"><code>OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME</code></p>
timeout	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 (continued)

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

pgrp	Pointer to the desired event flag group.
flags	<p>If opt (see below) is <code>OS_FLAG_SET</code>, each bit that is set in flags will set the corresponding bit in the event flag group. E.g., to set bits 0, 4 and 5 you would set flags to:</p> <p style="padding-left: 40px;"><code>0x31</code> (note, bit 0 is least significant bit)</p> <p>If opt (see below) is <code>OS_FLAG_CLR</code>, each bit that is set in flags will CLEAR the corresponding bit in the event flag group. E.g., to clear bits 0, 4 and 5 you would specify flags as:</p> <p style="padding-left: 40px;"><code>0x31</code> (note, bit 0 is least significant bit)</p>
opt	<p>Indicates whether the flags will be:</p> <p>set (<code>OS_FLAG_SET</code>), or cleared (<code>OS_FLAG_CLR</code>)</p>
err	<p>Pointer to an error code. Valid values are:</p> <ul style="list-style-type: none">• <code>OS_NO_ERR</code> - The call was successful.• <code>OS_FLAG_INVALID_PGRP</code> - null pointer passed.• <code>OS_ERR_EVENT_TYPE</code> - Not pointing to an event flag group• <code>OS_FLAG_INVALID_OPT</code> - Invalid option specified.

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

pgrp	Pointer to the desired event flag group.
err	Pointer to an error code returned to the called: <ul style="list-style-type: none">• OS_NO_ERR - The call was successful• OS_FLAG_INVALID_PGRP - null pointer passed.• OS_ERR_EVENT_TYPE - Not pointing to an event flag group

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 μ C/OS-II data; must be called before any other μ 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

pevent	Pointer to the event control block associated with the desired mailbox.
opt	May be one of the following delete options: <ul style="list-style-type: none">• <code>OS_DEL_NO_PEND</code> - Delete mailbox only if no task pending• <code>OS_DEL_ALWAYS</code> - Deletes the mailbox even if tasks are waiting. In this case, all the tasks pending will be readied.
err	Pointer to an error code that can contain one of the following values: <ul style="list-style-type: none">• <code>OS_NO_ERR</code> - Call was successful; mailbox was deleted• <code>OS_ERR_DEL_ISR</code> - Attempt to delete mailbox from ISR• <code>OS_ERR_INVALID_OPT</code> - Invalid option was specified• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the mailbox• <code>OS_ERR_EVENT_TYPE</code> - No pointer passed to a mailbox• <code>OS_ERR_PEVENT_NULL</code> - If <code>pevent</code> is a null pointer.

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

pevent	Pointer to mailbox's event control block.
timeout	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.
err	<p>Pointer to a variable for holding an error code. Possible error messages are:</p> <ul style="list-style-type: none">• OS_NO_ERR: The call was successful and the task received a message.• OS_TIMEOUT: A message was not received within the specified timeout• OS_ERR_EVENT_TYPE: Invalid event type• OS_ERR_PEND_ISR If this function was called from an ISR and the result would lead to a suspension.• OS_ERR_PEVENT_NULL: If pevent is a null pointer

RETURN VALUE

!= (void *)0	A pointer to the message received
== (void *)0	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

pevent	Pointer to mailbox's event control block.
msg	Pointer to message to be posted. A null pointer must not be sent.

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent.
OS_MBOX_FULL	The mailbox already contains a message. Only one message at a time can be sent and thus, the message MUST be consumed before another can be sent.
OS_ERR_EVENT_TYPE	Attempting to post to a non-mailbox.
OS_ERR_PEVENT_NULL	If pevent is a null pointer
OS_ERR_POST_NULL_PTR	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

pevent	Pointer to mailbox’s event control block.
msg	Pointer to the message to send. A null pointer must not be sent.
opt	Determines the type of POST performed: <ul style="list-style-type: none">• OS_POST_OPT_NONE - POST to a single waiting task (Identical to OS_MboxPost ())• OS_POST_OPT_BROADCAST - POST to ALL tasks that are waiting on the mailbox

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent.
OS_MBOX_FULL	The mailbox already contains a message. Only one message at a time can be sent and thus, the message MUST be consumed before another can be sent.
OS_ERR_EVENT_TYPE	Attempting to post to a non-mailbox.
OS_ERR_PEVENT_NULL	If pevent is a null pointer
OS_ERR_POST_NULL_PTR	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

pevent	Pointer to message mailbox's event control block.
pdata	Pointer to a data structure for information about the message mailbox

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent.
OS_ERR_EVENT_TYPE	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 μ C/OS-II.

PARAMETERS

addr	Pointer to starting address of the partition.
nblks	Number of memory blocks to create in the partition.
blksize	The size (in bytes) of the memory blocks.
err	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

pmem	Pointer to partition's memory control block
err	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

pmem	Pointer to the partition's memory control block.
pblk	Pointer to the memory block being released.

RETURN VALUE

OS_NO_ERR	The memory block was inserted into the partition.
OS_MEM_FULL	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

pmem	Pointer to partition's memory control block.
pdata	Pointer to structure for holding information about the partition.

RETURN VALUE

OS_NO_ERR	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: <ul style="list-style-type: none">• <code>OS_NO_ERR</code> - if the call was successful.• <code>OS_ERR_EVENT_TYPE</code> - if <code>pevent</code> is not a pointer to a mutex• <code>OS_ERR_PEVENT_NULL</code> - <code>pevent</code> is a null pointer• <code>OS_ERR_PEND_ISR</code> - 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

prio	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.
err	Pointer to error code that will be returned to your application: <ul style="list-style-type: none">• OS_NO_ERR - if the call was successful.• OS_ERR_CREATE_ISR - you attempted to create a mutex from an ISR• OS_PRIO_EXIST - a task at the priority inheritance priority already exist.• OS_ERR_PEVENT_NULL - no more event control blocks available.• OS_PRIO_INVALID - if the priority you specify is higher than the maximum allowed (i.e. > OS_LOWEST_PRIO)

RETURN VALUE

!= (void *)0	Pointer to the event control clock (OS_EVENT) associated with the created mutex.
== (void *)0	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

pevent	Pointer to mutex's event control block.
opt	May be one of the following delete options: <ul style="list-style-type: none">• <code>OS_DEL_NO_PEND</code> - Delete mutex only if no task pending• <code>OS_DEL_ALWAYS</code> - Deletes the mutex even if tasks are waiting. In this case, all pending tasks will be readied.
err	Pointer to an error code that can contain one of the following values: <ul style="list-style-type: none">• <code>OS_NO_ERR</code> - The call was successful and the mutex was deleted• <code>OS_ERR_DEL_ISR</code> - Attempted to delete the mutex from an ISR• <code>OS_ERR_INVALID_OPT</code> - An invalid option was specified• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the mutex• <code>OS_ERR_EVENT_TYPE</code> - If you didn't pass a pointer to a mutex pointer.

RETURN VALUE

pevent	On error.
(OS_EVENT *) 0	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

pevent	Pointer to mutex's event control block.
timeout	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.
err	<p>Pointer to where an error message will be deposited. Possible error messages are:</p> <p>OS_NO_ERR - The call was successful and your task owns the mutex</p> <p>OS_TIMEOUT - The mutex was not available within the specified time.</p> <p>OS_ERR_EVENT_TYPE - If you didn't pass a pointer to a mutex</p> <p>OS_ERR_PEVENT_NULL - pevent is a null pointer</p> <p>OS_ERR_PEND_ISR - If you called this function from an ISR and the result would lead to a suspension.</p>

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

OS_NO_ERR	The call was successful and the mutex was signaled.
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_POST_ISR	Attempted to post from an ISR (invalid for mutexes)
OS_ERR_NOT_MUTEX_OWNER	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

pevent	Pointer to the event control block associated with the desired mutex.
pdata	Pointer to a structure that will contain information about the mutex.

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent
OS_ERR_QUERY_ISR	Function was called from an ISR
OS_ERR_PEVENT_NULL	pevent is a null pointer
OS_ERR_EVENT_TYPE	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

start	Pointer to the base address of the message queue storage area. The storage area MUST be declared an array of pointers to void: void *MessageStorage[qsize] .
qsize	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

pevent	Pointer to the queue's event control block.
opt	May be one of the following delete options: <ul style="list-style-type: none">• <code>OS_DEL_NO_PEND</code> - Delete queue only if no task pending• <code>OS_DEL_ALWAYS</code> - Deletes the queue even if tasks are waiting. In this case, all the tasks pending will be readied.
err	Pointer to an error code that can contain one of the following: <ul style="list-style-type: none">• <code>OS_NO_ERR</code> - Call was successful and queue was deleted• <code>OS_ERR_DEL_ISR</code> - Attempt to delete queue from an ISR• <code>OS_ERR_INVALID_OPT</code> - Invalid option was specified• <code>OS_ERR_TASK_WAITING</code> - One or more tasks were waiting on the queue• <code>OS_ERR_EVENT_TYPE</code> - You didn't pass a pointer to a queue• <code>OS_ERR_PEVENT_NULL</code> - If <code>pevent</code> is a null pointer.

RETURN VALUE

pevent	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

pevent	Pointer to message queue's event control block.
timeout	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.
err	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

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	If a pointer to a queue not passed.
OS_ERR_PEVENT_NULL	If pevent is a null pointer.
OS_ERR_POST_NULL_PTR	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 <code>pevent</code> 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

pevent	Pointer to message queue’s event control block.
msg	Pointer to the message to send. A null pointer must not be sent.
opt	Determines the type of POST performed: <ul style="list-style-type: none">• <code>OS_POST_OPT_NONE</code> - POST to a single waiting task (Identical to <code>OSQPost()</code>)• <code>OS_POST_OPT_BROADCAST</code> - POST to ALL tasks that are waiting on the queue• <code>OS_POST_OPT_FRONT</code> - POST as LIFO (Simulates <code>OSQPostFront()</code>) The last 2 flags may be combined: <ul style="list-style-type: none">• <code>OS_POST_OPT_FRONT + OS_POST_OPT_BROADCAST</code> - is identical to <code>OSQPostFront()</code> except that it will broadcast msg to all waiting tasks.

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

pevent	Pointer to message queue's event control block.
pdata	Pointer to a data structure for message queue information.

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent
OS_ERR_EVENT_TYPE	Attempting to obtain data from a non queue.
OS_ERR_PEVENT_NULL	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 OSMemAccept( OS_EVENT *pevent );
```

DESCRIPTION

This function checks the semaphore to see if a resource is available or if an event occurred. Unlike OSMemPend(), OSMemAccept() 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

OSMemCreate, OSMemPend, OSMemPost, OSMemQuery

OSSemCreate

```
OS_EVENT *OSSemCreate( 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, OSSemPend, OSSemPost, OSSemQuery

OSSemPend

```
void OSMemPend( 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

OSMemAccept, OSMemCreate, OSMemPost, OSMemQuery

OSSemPost

```
INT8U OSMemPost ( 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 <code>OSMemAccept ()</code> or <code>OSMemPend ()</code> .
OS_ERR_EVENT_TYPE	If a pointer to a semaphore not passed.
OS_ERR_PEVENT_NULL	If <code>pevent</code> is a null pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

`OSMemAccept`, `OSMemCreate`, `OSMemPend`, `OSMemQuery`

OSSemQuery

```
INT8U OSMemQuery( OS_EVENT *pevent, OS_SEM_DATA *pdata );
```

DESCRIPTION

Obtains information about a semaphore.

PARAMETERS

pevent	Pointer to the desired semaphore's event control block
pdata	Pointer to a data structure that will hold information about the semaphore.

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent.
OS_ERR_EVENT_TYPE	Attempting to obtain data from a non semaphore.
OS_ERR_PEVENT_NULL	If the <code>pevent</code> parameter is a null pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMemAccept, OSMemCreate, OSMemPend, OSMemPost

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 μ 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

oldprio	The priority level to change from.
newprio	The priority level to change to.

RETURN VALUE

OS_NO_ERR	The call was successful.
OS_PRIO_INVALID	The priority specified is higher than the maximum allowed (i.e. \geq OS_LOWEST_PRIO).
OS_PRIO_EXIST	The new priority already exist
OS_PRIO_ERR	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 μ 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

task	Pointer to the task's starting address.
pdata	Pointer to a task's initial parameters.
stk_size	Number of bytes of the stack.
prior	The task's unique priority number.

RETURN VALUE

OS_NO_ERR	The call was successful.
OS_PRIO_EXIT	Task priority already exists (each task MUST have a unique priority).
OS_PRIO_INVALID	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 μ 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

task	Pointer to task's code.
pdata	Pointer to optional data area; used to pass parameters to the task at start of execution.
prio	The task's unique priority number; the lower the number the higher the priority.
id	The task's identification number (0...65535).
stk_size	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.
pext	Pointer to a user-supplied Task Control Block (TCB) extension.
opt	The lower 8 bits are reserved by μ C/OS-II. The upper 8 bits control application-specific options. Select an option by setting the corresponding bit(s).

RETURN VALUE

OS_NO_ERR	The call was successful.
OS_PRIO_EXIT	Task priority already exists (each task MUST have a unique priority).
OS_PRIO_INVALID	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 μ 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

OS_NO_ERR	The call was successful.
OS_TASK_DEL_IDLE	Attempting to delete μ C/OS-II's idle task.
OS_PRIO_INVALID	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.
OS_TASK_DEL_ERR	The task to delete does not exist.
OS_TASK_DEL_ISR	Attempting to delete a task from an ISR.

LIBRARY

`UCOS2.LIB`

SEE ALSO

`OSTaskDelReq`

OSTaskDelHook

```
void OSTaskDelHook( OS_TCB *ptcb );
```

DESCRIPTION

Called by μ 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

OS_NO_ERR	The task exists and the request has been registered.
OS_TASK_NOT_EXIST	The task has been deleted. This allows the caller to know whether the request has been executed.
OS_TASK_DEL_IDLE	If requesting to delete uC/OS-II's idletask.
OS_PRIO_INVALID	The priority specified is higher than the maximum allowed (i.e. \geq OS_LOWEST_PRIO) or, OS_PRIO_SELF is not specified.
OS_TASK_DEL_REQ	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

prio	Priority number of the task.
pdata	Pointer to task's TCB.

RETURN VALUE

OS_NO_ERR	The requested task is suspended.
OS_PRIO_INVALID	The priority you specify is higher than the maximum allowed (i.e. \geq OS_LOWEST_PRIO) or, OS_PRIO_SELF is not specified.
OS_PRIO_ERR	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 μ 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

prio	The task's priority.
pdata	Pointer to a data structure of type OS_STK_DATA.

RETURN VALUE

OS_NO_ERR	The call was successful.
OS_PRIO_INVALID	The priority you specify is higher than the maximum allowed (i.e. > OS_LOWEST_PRIO) or, OS_PRIO_SELF not specified.
OS_TASK_NOT_EXIST	The desired task has not been created.
OS_TASK_OPT_ERR	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

OS_NO_ERR	The requested task is suspended.
OS_TASK_SUS_IDLE	Attempting to suspend the idle task (not allowed).
OS_PRIO_INVALID	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.
OS_TASK_SUS_PRIO	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

hours	Number of hours that the task will be delayed (max. is 255)
minutes	Number of minutes (max. 59)
seconds	Number of seconds (max. 59)
milli	Number of milliseconds (max. 999)

RETURN VALUE

OS_NO_ERR	Execution delay of task was successful
OS_TIME_INVALID_MINUTES	Minutes parameter out of range
OS_TIME_INVALID_SECONDS	Seconds parameter out of range
OS_TIME_INVALID_MS	Milliseconds parameter out of range
OS_TIME_ZERO_DLY	

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

OS_NO_ERR	Task has been resumed.
OS_PRIO_INVALID	The priority you specify is higher than the maximum allowed (i.e. \geq <code>OS_LOWEST_PRIO</code>).
OS_TIME_NOT_DLY	Task is not waiting for time to expire.
OS_TASK_NOT_EXIST	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 μ C/OS-II. The returned value corresponds to μ 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

outchr

```
char outchr(char c, int n, int (*putc) () );
```

DESCRIPTION

Use `putc` to output `n` times the character `c`.

PARAMETERS

c	Character to output
n	Number of times to output
putc	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. Use caution when converting address in the E000-FFFF range. Returns the 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 <code>pool_init()</code> . 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 <code>pool_init()</code> or <code>pool_xinit()</code> .
----------	--

RETURN VALUE

Number of elements available for allocation.

LIBRARY

POOL.LIB

SEE ALSO

`pool_init`, `pool_xinit`, `phwm`, `pncl`

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 <code>pool_init()</code> or <code>pool_xinit()</code> . 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`

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.

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 <code>pool_init()</code> . 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

p	Pool handle structure, as previously passed to <code>palloc()</code> .
e	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

p	Pool handle structure, as previously passed to <code>pool_alloc()</code> or <code>palloc_fast</code> . This must be in the IX register.
e	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 <code>pool_init()</code> or <code>pool_xinit()</code> .
----------	--

RETURN VALUE

Maximum number of elements ever allocated.

LIBRARY

POOL.LIB

SEE ALSO

`pool_init`, `pool_xinit`, `pavail`

pktXclose

```
void pktXclose( void );
```

DESCRIPTION

Disables serial port X, where X is A|B|C|D. Starting with Dynamic C version 7.25, the functions `pktEclose()` and `pktFclose()` may be used with the Rabbit 3000 microprocessor.

LIBRARY

PACKET.LIB

pktXgetErrors

```
char pktXgetErrors( void );
```

DESCRIPTION

Gets a bit field with flags set for any errors that occurred on port X, where X is A|B|C|D. These flags are then cleared, so that a particular error will only cause the flag to be set once.

Starting with Dynamic C version 7.25, the functions `pktEgetErrors()` and `pktFgetErrors()` may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

A bit field with flags for various errors. The errors along with their bit masks are as follows:

PKT_BUFFEROVERFLOW	0x01
PKT_RXOVERRUN	0x02
PKT_PARITYERROR	0x04
PKT_NOBUFFER	0x08

LIBRARY

PACKET.LIB

pktXinitBuffers

```
int pktXinitBuffers( int buf_count, int buf_size ); X = A|B|C|D
```

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.

Starting with Dynamic C version 7.25, the functions `pktEinitBuffers()` and `pktFinitBuffers()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

buf_count	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.
buf_size	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)() );
```

DESCRIPTION

Opens serial port X, where X is A|B|C|D. Starting with Dynamic C version 7.25, the functions `pktEopen()` and `pktFopen()` may be used with the Rabbit 3000 microprocessor.

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 (continued)

PARAMETERS

baud	Bits per second of data transfer: minimum is 2400.
mode	Type of packet scheme used, the options are: <ul style="list-style-type: none">• PKT_GAPMODE• PKT_9BITMODE• PKT_CHARMODE
options	Further specification for the packet scheme. The value of this depends on the mode used: <ul style="list-style-type: none">• gap mode - minimum gap size (in byte times)• 9-bit mode - type of 9-bit protocol<ul style="list-style-type: none">• 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 );
```

DESCRIPTION

Gets a received packet, if there is one, from serial port X, where X is A|B|C|D. Starting with Dynamic C version 7.25, the functions `pktEreceive()` and `pktFreceive()` may be used with the Rabbit 3000 microprocessor.

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 );
```

DESCRIPTION

Initiates the sending of a packet of data using serial port X, where X is A|B|C|D. 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.

Starting with Dynamic C version 7.25, the functions `pktEsend()` and `pktFsend()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

send_buffer	The data to be sent
buffer_length	Length of the data buffer to transmit
delay	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();
```

DESCRIPTION

Tests if a packet is currently being sent on serial port X, where X=A|B|C|D. If `pktXsending()` returns true, the transmitter is busy and cannot accept another packet.

Starting with Dynamic C version 7.25, the functions `pktEsending()` and `pktFsending()` may be used with the Rabbit 3000 microprocessor.

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 );
```

DESCRIPTION

Configures parity generation and checking. Can also configure for 2 stop bits.

Starting with Dynamic C version 7.25, the functions `pktEsetParity()` and `pktFsetParity()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

- | | |
|-------------|---|
| mode | Code for mode of parity bit: <ul style="list-style-type: none">• PKT_NOPARITY - no parity bit (8N1 format, default)• PKT_OPARITY - odd parity (8O1 format)• PKT_EPARITY - even parity (8E1 format)• PKT_TWOSTOP - an extra stop bit (8N2 format) |
|-------------|---|

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 <code>pool_init()</code> . 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

p	Pool handle structure, as previously passed to <code>pool_init()</code> .
e	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). 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.
d	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 (continued)

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 el's
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

p	Pool handle structure, as previously passed to <code>pool_init()</code> . Pass in IX register
e	Address of element to move. Pass in DE register.
d	The first reference element. Pass in BC register.
f	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 <code>pool_init()</code> or <code>pool_xinit()</code> .
----------	--

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

p	Pool handle structure, as previously passed to <code>pool_init()</code> .
e	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. 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

p	Pool handle structure, as previously passed to <code>pool_init()</code> . Pass this in IX register.
e	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

x	Variable of the polynomial.
n	The order of the polynomial
c	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

p	Pool handle structure, as previously passed to <code>pool_init()</code> .
base	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.
nel	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

p	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> .
base	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.
nel	Number of elements in the memory area. 1..32767
elsize	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

- | | |
|-------------|---|
| p | Pool handle structure, as previously passed to <code>pool_init()</code> or <code>pool_xinit()</code> . |
| link | Must be one of the following: <ul style="list-style-type: none">• <code>POOL_NOT_LINKED (0)</code>: the pool is not to be linked.• <code>POOL_LINKED_AUTO (1)</code>: the pool is linked, and newly allocated elements are always added at the end of the list.• <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. |

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

p	Pool handle structure, as previously passed to <code>pool_xinit()</code> .
base	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> .
nel	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

p	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.
base	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.
nel	Number of elements in the memory area. 1..65535
elsize	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

x	Value to be raised
y	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

x	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 f_{max} term and will affect the calculation of the dc power. If the dc power or the f_{max} power is important, the f_{max} 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

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

LIBRARY

FFT.LIB

SEE ALSO

`fftcplx`, `fftcplxinv`, `fftrealm`, `fftrealmv`, `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 <code>pool_init()</code> .
e	Previous element address, obtained by, e.g., <code>plast()</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. If this parameter is null, then the result is the same as <code>plast()</code> . This ensures the invariant

`pprev(p, 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

p	Pool handle structure, as previously passed to <code>pool_init()</code> . Pass this in IX register.
e	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`

premain

```
void premain();
```

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

p	Pool handle structure, as previously passed to <code>pool_init()</code> .
e	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).
where	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).
options	Option flags. Currently, the only options are: <div style="margin-left: 40px;"><code>POOL_INSERT_BEFORE</code> <code>POOL_INSERT_AFTER</code></div> which specifies whether “e” is to be inserted before or after “where.”

preorder (continued)

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

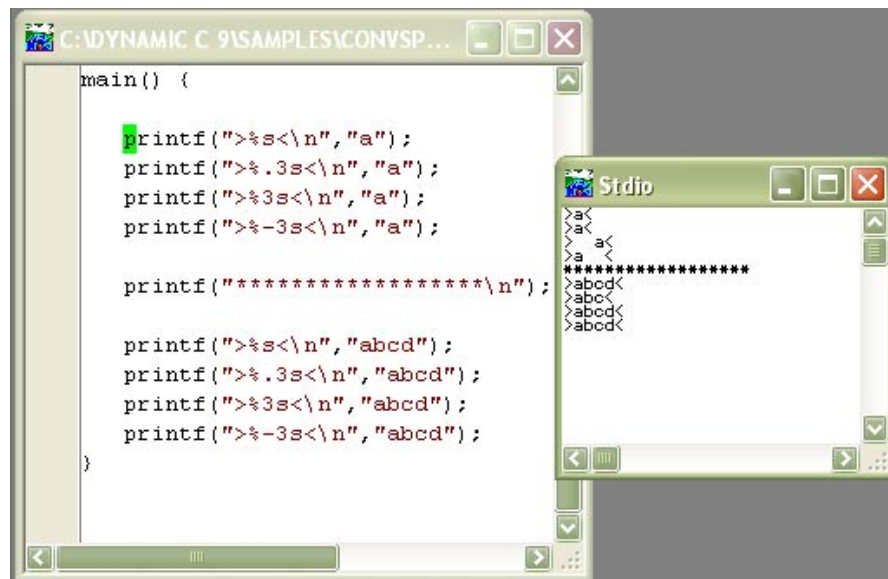
`sprintf`

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.



```
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");  
}
```

The screenshot shows a C program in a text editor. The program uses `printf` to demonstrate various string formatting specifiers. The output window shows the results: the first four lines show the string "a" formatted with different precision and width specifiers, followed by a separator line of asterisks, and then the string "abcd" formatted with different precision and width specifiers.

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.

%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 **'-'** 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 **'-'** 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
```

%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
```

%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
```

%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 microprocessor.

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

R3000.LIB

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 microprocessor.

PARAMETERS

channel	channel(0 to 3)
duty_cycle	value from 0 to 1024
options	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

R3000.LIB

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 <code>pool_init()</code> . 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 <code>pool_init()</code> . 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

p	Pool handle structure, as previously passed to <code>pxalloc()</code> .
e	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

p	Pool handle structure, as previously passed to <code>palloc()</code> or <code>palloc_fast</code> . This must be in the IX register.
e	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 <code>pool_xinit()</code> . 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 <code>pool_xinit()</code> .
e	Previous element address, obtained by e.g. <code>pxfirst()</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>pxfirst()</code> . This ensures the invariant <code>pxnext(p, pxprev(p, e)) == e.</code>

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

p	Pool handle structure, as previously passed to <code>pool_xinit()</code> . Pass this in the IX register.
e	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

p	Pool handle structure, as previously passed to <code>pool_xinit()</code> .
e	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

p	Pool handle structure, as previously passed to <code>pool_xinit()</code> . Pass this in IX register.
e	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 microprocessor. prominent

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

R3000.LIB

qd_init

```
void qd_init( int iplevel );
```

DESCRIPTION

Initializes the quadrature decoders and sets up the ISR. This must be called before any other QD functions are used. Sets up the lower nibble of port F to be the QD input pins.

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

iplevel	The interrupt priority for the ISR that handles the count overflow. This should usually be 1.
----------------	---

LIBRARY

R3000.LIB

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 microprocessor.

PARAMETERS

channel	The channel to read (currently 1 or 2).
----------------	---

RETURN VALUE

Returns a signed long for the current count.

LIBRARY

R3000.LIB

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 microprocessor.

PARAMETERS

channel The channel to reset (currently 1 or 2)

LIBRARY

R3000.LIB

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

base	Base address of the raw string data.
n	Number of blocks to sort.
s	Number of bytes in each block.
cmp	User-supplied compare routine for two block pointers, p and q , that returns an int with the same rules used by Unix <code>strcmp</code> (p , q): = 0: Blocks p and q are equal < 0: p is less than q > 0: p is greater than q 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 (2π 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.

PARAMETERS

port Address of internal 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

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

input	Input bit file.
buf	Output buffer.
lenx	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.

PARAMETERS

dest	Pointer to destination to copy data to.
addr	Address offset in User block to read from.
numbytes	Number of bytes to copy.

RETURN VALUE

0: Success
-1: Invalid address or range

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.

PARAMETERS

dests	Pointer to array of destinations to copy data to.
numbytes	Array of numbers of bytes to be written to each destination.
numdests	Number of destinations.
addr	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)

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();
```

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

root2xmem

```
int root2xmem( unsigned long dest, void *src, unsigned len );
```

DESCRIPTION

Stores `len` characters from logical address `src` to physical address `dest`.

PARAMETERS

dest	Physical address.
src	Logical address.
len	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

seconds	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.
tzname	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();
```

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`

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

rx_byte	The 8 bit character being tested for parity.
parity	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 = A|B|C|D|E|F */
```

DESCRIPTION

Disables serial port X. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEclose()` and `serFclose()` may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXdatabits

```
void serXdatabits ( state ); /* where X = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `serEdatabits()` and `serFdatabits()` may be used with the Rabbit 3000 microprocessor.

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 seven bit data.
PARAM_8BIT	Configures serial port to use eight bit data (default).

LIBRARY

RS232.LIB

serXflowcontrolOff

```
void serXflowcontrolOff(); /* where X = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `serEflowcontrolOff()` and `serFflowcontrolOff()` may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXflowcontrolOn

```
void serXflowcontrolOn(); /* where X = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `serEflowcontrolOn()` and `serFflowcontrolOn()` may be used with the Rabbit 3000 microprocessor.

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.

`SERX_RTS_PORT` Data register for the parallel port that the RTS line is on. e.g.
 PCDR

`SERA_RTS_SHADOW` Shadow register for the RTS line's parallel port. e.g.
 PCDRShadow

`SERA_RTS_BIT` The bit number for the RTS line

`SERA_CTS_PORT` Data register for the parallel port that the CTS line is on

`SERA_CTS_BIT` The bit number for the CTS line

LIBRARY

`RS232.LIB`

serXgetc

```
int serXgetc (); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Get next available character from serial port X read buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEgetc()` and `serFgetc()` may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

Success: the next character in the low byte, 0 in the high byte.

Failure: -1

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes characters
main() {
    int c;
    serAopen(19200);
    while (1) {
        if ((c = serAgetc()) != -1) {
            serAputc(c);
        }
    }
    serAclose()
}
```

serXgetError

```
int serXgetError (); /* where X = A | B | C | D | E | 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

Starting with Dynamic C version 7.25, the functions `serEgetError()` and `serFgetError()` may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

The error flags byte.

LIBRARY

RS232.LIB

serXopen

```
int serXopen ( long baud ); /* where X = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `serEopen()` and `serFopen()` may be used with the Rabbit 3000 microprocessor.

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.

PARAMETERS

baud	Bits per second 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 baud rate achieved on the Rabbit is the same as the input baud rate. The software was able to calculate a valid divisor for the requested baud rate within 5%.
- 0: The baud rate achieved on the Rabbit does not match the input baud rate.

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 = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `serEparity()` and `serFparity()` may be used with the Rabbit 3000 microprocessor.

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` - Configures serial port to check/generate for odd parity.
- `PARAM_EPARITY` - Configures serial port to check/generate for even parity.
- `PARAM_2STOP` - Configures serial port to generate 2 stop bits.

LIBRARY

`RS232.LIB`

serXpeek

```
int serXpeek(); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Returns 1st character in input buffer X, without removing it from the buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEpeek()` and `serFpeek()` may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

An integer with 1st character in buffer in the low byte.
-1 if the buffer is empty.

LIBRARY

RS232.LIB

serXputc

```
int serXputc( char c ); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Writes a character to serial port X write buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEputc()` and `serFputc()` may be used with the Rabbit 3000 microprocessor.

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 = A|B|C|D|E|F */
```

DESCRIPTION

Calls `serXwrite(s, strlen(s))`. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEputs()` and `serFputs()` may be used with the Rabbit 3000 microprocessor.

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 char s[] = "Hello Z-World";
    serAopen(19200);
    while (1) {
        serAputs(s);
    }
    serAclose();
}
```

serXrdFlush

```
void serXrdFlush(); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Flushes serial port X input buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serErdFlush()` and `serFrdfFlush()` may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXrdFree

```
int serXrdFree(); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Calculates the number of characters of unused data space. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serErdFree()` and `serFrdfFree()` may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

The number of chars it would take to fill input buffer X.

LIBRARY

RS232.LIB

serXrdUsed

```
int serXrdUsed(); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Calculates the number of characters ready to read from the serial port receive buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serErUsed()` and `serFrUsed()` may be used with the Rabbit 3000 microprocessor.

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 = A|B|C|D|E|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.

Starting with Dynamic C version 7.25, the functions `serEread()` and `serFread()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure to read from serial port <code>X</code>
length	Number of bytes to read
tmout	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(); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Flushes serial port X transmit buffer, meaning that the buffer contents will not be sent. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEwrFlush()` and `serFwrFlush()` may be used with the Rabbit 3000 microprocessor.

LIBRARY

`RS232.LIB`

serXwrFree

```
int serXwrfree(); /* where X = A|B|C|D|E|F */
```

DESCRIPTION

Calculates the free space in the serial port transmit buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEwrFree()` and `serFwrFree()` may be used with the Rabbit 3000 microprocessor.

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 = A|B|C|D|E|F */
```

DESCRIPTION

Transmits `length` bytes to serial port X. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions `serEwrite()` and `serFwrite()` may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure to write to serial port X
length	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 Z-World";
    serAopen(19200);
    while (1) {
        serAwrite(s, strlen(s));
    }
    serAclose();
}
```

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

address	Address of byte containing bits 7-0
bit	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

address Address of byte containing bits 7-0.

bit 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: <ul style="list-style-type: none">• 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:
	<ul style="list-style-type: none">• 0 = no modulation• 1 = weak modulation• 2 = strong modulation

LIBRARY

SYS.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()`.

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:                // like 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`

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 I0CR for external interrupt 0 and I1CR for external interrupt 1.

The actual priority used by the interrupt service routine (ISR) is passed to this function.

PARAMETERS

priority	Priority the ISR should run at. Valid values are 1, 2 or 3.
isr	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`

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` argument that should be used for each peripheral or RST. The offset into the vector table is also shown. The following vectors are for the Rabbit 2000 and 3000.

Peripheral or RST	vectNum	Vector Table Offset
System Management (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

SetVectIntern (continued)

The following vectors are for the Rabbit 3000 processor only.

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

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();
```

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

dev	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.
cs_port	I/O port for the active low chip select pin for the device.
cs_shadow	Pointer to the shadow variable for <code>cs_port</code> .
cs_pin	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

dev	Pointer to <code>sf_device</code> struct for initialized flash device.
buffer	Address of an xmem buffer.
offset	The address in the serial flash RAM to start reading from.
len	The number of bytes to read.
flags	Can be one of the following: SF_BITSREVERSED - 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>) SF_RAMBANK1(default) - Reads from the first RAM bank on the flash device SF_RAMBANK2 - 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

dev	Pointer to <code>sf_device</code> struct for initialized flash device.
bank	Which RAM bank to write the data to. For Atmel 45DBxxx devices, this can be 1 or 2.
page	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

buffer	Pointer to character buffer to copy data into.
offset	Address in the serial flash RAM to start reading from
len	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

dev	Pointer to sf_device struct for initialized flash device.
buffer	Pointer to xmem data to write into the flash chip RAM.
offset	The address in the serial flash RAM to start writing at.
len	The number of bytes to write.
flags	Can be one of the following: <ul style="list-style-type: none">• SF_BITSREVERSED - 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 sf_XReadRAM)• SF_RAMBANK1 (default) - Writes to the first RAM bank on the flash device• SF_RAMBANK2 - Writes to the alternate RAM bank on the flash device

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

dev	Pointer to <code>sf_device</code> struct for initialized flash device.
bank	Which RAM bank to write the data from. For Atmel 45DBxxx devices, this can be 1 or 2
page	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

buffer	Pointer to data that will be written the flash chip RAM.
offset	Address in the serial flash RAM to start writing at.
len	Number of bytes to write.

RETURN VALUE

0 for success
-1 for error

LIBRARY

SFLASH.LIB

sfspi_init

```
int sfspi_init()
```

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 function 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, "%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

buffer	Location of formatted string.
len	The maximum length of the formatted string.
format	String to be formatted.
...	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 ();
```

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

DestAddr	Address to store the data
ByteCount	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

SrcAddr	Address of data to write.
ByteCount	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

SrcAddr	Address of data to write.
DstAddr	Address to put received data.
ByteCount	Number of bytes to read/write. The maximum value is 255 bytes. This limit is not checked! The receive buffer MUST 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,

```
sprintf(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

buffer	Result string of the formatted string.
format	String to be formatted.
...	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

```
char *strcat( char *dst, char *src );
```

DESCRIPTION

Appends one string to another.

PARAMETERS

dst	Pointer to location to destination string.
src	Pointer to location to source string.

RETURN VALUE

Pointer to destination string.

LIBRARY

STRING.LIB

SEE ALSO

strncat

strchr

```
char *strchr( char *src, char ch );
```

DESCRIPTION

Scans a string for the first occurrence of a given character.

PARAMETERS

src	String to be scanned.
ch	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

```
int strcmp( char *str1, char *str2 )
```

DESCRIPTION

Performs unsigned character by character comparison of two null terminated strings.

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

```
int *strcmpi( char *str1, char *str2 );
```

DESCRIPTION

Performs case-insensitive unsigned character by character comparison of two null terminated strings.

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

strncmpi, strncmp, strcmp

strcpy

```
char *strcpy( char *dst, char *src );
```

DESCRIPTION

Copies one string into another string including the null terminator.

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

```
unsigned int strcspn( char *s1, char *s2 );
```

DESCRIPTION

Scans a string for the occurrence of any of the characters in another string.

PARAMETERS

s1	String to be scanned.
s2	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

```
int strlen( char *s );
```

DESCRIPTION

Calculate the length of a string.

PARAMETERS

s Character string.

RETURN VALUE

Number of bytes in a string.

LIBRARY

STRING.LIB

strncat

```
char *strncat( char *dst, char *src, unsigned int n );
```

DESCRIPTION

Appends one string to another up to and including the null terminator or until *n* characters are transferred, followed by a null terminator.

PARAMETERS

dst Pointer to location to receive string.

src Pointer to location to supply string.

n 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

```
int strncmp( char *str1, char *str2, n )
```

DESCRIPTION

Performs unsigned character by character comparison of two strings of length n.

PARAMETERS

str1	Pointer to string 1.
str2	Pointer to string 2.
n	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

```
int strncmpi( char *str1, char *str2, unsigned n )
```

DESCRIPTION

Performs case-insensitive unsigned character by character comparison of two strings of length `n`.

PARAMETERS

str1	Pointer to string 1.
str2	Pointer to string 2.
n	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

```
char *strncpy( char *dst, char *src, unsigned int n );
```

DESCRIPTION

Copies a given number of characters from one string to another and padding with null characters or truncating as necessary.

PARAMETERS

dst	Pointer to location to receive string.
src	Pointer to location to supply string.
n	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

```
char *strpbrk( char *s1, char *s2 );
```

DESCRIPTION

Scans a string for the first occurrence of any character from another string.

PARAMETERS

s1	String to be scanned.
s2	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

```
char *strrchr( char *s, int c );
```

DESCRIPTION

Similar to `strchr`, except this function searches backward from the end of `s` to the beginning.

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

```
size_t strspn( char *src, char *brk );
```

DESCRIPTION

Scans a string for the first segment in `src` containing only characters specified in `brk`.

PARAMETERS

<code>src</code>	String to be scanned
<code>brk</code>	Set of characters

RETURN VALUE

Returns the length of the segment.

LIBRARY

`STRING.LIB`

strstr

```
char *strstr( char *s1, char *s2 );
```

DESCRIPTION

Finds a substring specified by `s2` in string `s1`.

PARAMETERS

s1	String to be scanned.
s2	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`, `strrchr`, `strtok`

strtod

```
float strtod( char *s, char **tailptr );
```

DESCRIPTION

ANSI string to float conversion.

PARAMETERS

s	String to convert.
tailptr	Pointer to a pointer of character. The next conversion may resume at the location specified by *tailptr.

RETURN VALUE

The float number.

LIBRARY

STRING.LIB

SEE ALSO

atof

strtok

```
char *strtok( char *src, char *brk );
```

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

```
long strtol( char *sptr, char **tailptr, int base );
```

DESCRIPTION

ANSI string to long conversion.

PARAMETERS

sptr	String to convert.
tailptr	Assigned the last position of the conversion. The next conversion may resume at the location specified by *tailptr.
base	Indicates the radix of conversion.

RETURN VALUE

The long integer.

LIBRARY

STRING.LIB

SEE ALSO

atoi, atol

`_sysIsSoftReset`

```
void _sysIsSoftReset();
```

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° ($\text{PI}/2$) 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

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

mktime, 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 cooperative multitasking program or if the `SEC_TIMER` variable is being used in another way the user, so user beware.

PARAMETERS

t	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

`mktime`, `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();
```

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();
```

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();
```

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

setting

Divider setting. The following are valid:

- CLKDIV_2 - divide main processor clock by two
- CLKDIV_4 - divide main processor clock by four
- CLKDIV_6 - divide main processor clock by six
- CLKDIV_8 - divide main processor clock by eight

RETURN VALUE

None.

LIBRARY

SYS.LIB

SEE ALSO

`useClockDivider`, `useMainOsc`, `use32kHzOsc`, `set32kHzDivider`

useMainOsc

```
void useMainOsc();
```

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

value	16-bit number to convert.
buf	Character string of converted number.

RETURN VALUE

Pointer to null at end of string.

LIBRARY

`STDIO.LIB`

SEE ALSO

`itoa`, `htoa`, `ltoa`

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 <code>VdGetFreeWd()</code>
-------------	---

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

flashDst	Physical address of the flash destination.
rootSrc	Array of pointers to the root sources.
numbytes	Array of numbers of bytes to write for each source.
numsources	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

time	32-bit value representing the number of seconds since January 1, 1980.
-------------	--

LIBRARY

RTCLOCK.C

SEE ALSO

`read_rtc`

writeUserBlock

```
int writeUserBlock( unsigned addr, void *source, unsigned
    numbytes );
```

DESCRIPTION

Z-World 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. This is where calibration constants are stored for boards with analog I/O. Space here 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.

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. Z-World manufactured boards with large sector flash will have valid System and User ID blocks, so this should not be problem on Z-World 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 (continued)

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.

LIBRARY

IDBLOCK.LIB

SEE ALSO

readUserBlock, writeUserBlockArray

writeUserBlockArray

```
int writeUserBlockArray( unsigned addr, void* sources[], unsigned
    numbytes[], int numsources );
```

DESCRIPTION

Z-World 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 is 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.

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. Z-World manufactured boards with large sector flash will have valid System ID and User blocks, so is not a problem on Z-World 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.

PARAMETERS

addr	Address offset in User block to write to.
sources	Array of pointer to sources to copy data from.
numbytes	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.
numsources	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.

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

port	Address of external data register.
portshadow	Reference pointer to a variable shadowing the register data. Substitute with null pointer (or 0) if shadowing is not required.
data_value	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

port	Address of data register.
portshadow	Reference pointer to a variable shadowing the register data. Substitute with null pointer (or 0) if shadowing is not required.
data_value	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.

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 run-time error will occur.

PARAMETERS

- | | |
|--------------|---|
| sz | 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. |
| align | 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. |
| type | One of the following values: <ul style="list-style-type: none">• <code>XALLOC_ANY</code> (0) - any type of RAM storage allowed• <code>XALLOC_BB</code> (1) - must be battery-backed RAM. This is currently supported only on RCM3200 and derivations.• <code>XALLOC_NOTBB</code> (2) - return non-BB RAM only.• <code>XALLOC_MAYBBB</code> (3) - return non-BB RAM in preference to BB. Any other value has undefined results. |

RETURN VALUE

The 20-bit physical address of the allocated data on success. On error, a run-time 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 Reserved for future use. Set to 0.

LIBRARY

XMEM.LIB

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 <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 (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

<code>addr_ptr</code>	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> .
<code>align</code>	Alignment of returned block, as per <code>_xalloc()</code> .
<code>type</code>	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

data	Physical address of the 256 byte data buffer
old_ecc	Pointer to the old (original) 3 byte ECC's buffer
new_ecc	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

dest	Logical address
src	Physical address
len	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

dest	Physical address of destination
src	Physical address of source data
len	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

src	xmem (linear) address of the first character to search.
ch	Character to search for.
n	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

xstr	xmem (linear) address of the first character of the first string to compare.
str	root address of the first character of the second string to compare.
n	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

addr	Address of storage previously obtained by <code>_xalloc()</code> .
sz	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

dst	<code>xmem</code> (linear) address of the int value to set.
val	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

dst	<code>xmem</code> (linear) address of the float value to set.
val	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

dst	<code>xmem</code> (linear) address of the long integer value to set.
val	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

src xmem (linear) address of the first character of the string. Note: to perform a normal null-terminated search, ensure that `src` is in the range $0..2^{20}-1$. If the MSB of `src` 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

Dynamic C Function Reference Manual

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

©2004 Z-World Inc. • All rights reserved.

Z-World reserves the right to make changes and improvements to its products without providing notice.

Notice to Users

Z-WORLD PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS UNLESS A SPECIFIC WRITTEN AGREEMENT REGARDING SUCH INTENDED USE IS ENTERED INTO BETWEEN THE CUSTOMER AND Z-WORLD PRIOR TO USE. Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

Trademarks

Dynamic C[®] is a registered trademark of Z-World Inc.

Windows[®] is a registered trademark of Microsoft Corporation

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792
www.zworld.com

Z-World Software End User License Agreement

IMPORTANT-READ CAREFULLY: BY INSTALLING, COPYING OR OTHERWISE USING THE ENCLOSED Z-WORLD,INC. ("Z-WORLD") 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 Z-WORLD 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 Z-World 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 (uC/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 Z-WORLD, or (ii) based on the Rabbit 2000 microprocessor or the Rabbit 3000 microprocessor. Qualified Systems may not be (a) designed or intended to be re-programmable by your customer using the Software, or (b) competitive with Z-WORLD products, except as otherwise stated in a written agreement between Z-World and the system manufacturer. Such written agreement may require an end user to pay run time royalties to Z-World.

2. **License.** Z-WORLD 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 Z-Worlds 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 Z-WORLD; 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 Z-WORLD 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, Z-WORLD 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 Z-WORLD, 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 Z-WORLD, and will be governed solely by the terms agreed upon between You and the third party providing such services. Z-WORLD 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. Z-WORLD 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, Z-WORLD 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 Z-WORLD 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 Z-WORLD 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 Z-WORLD 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 Z-WORLDS 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 Z-WORLD 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 Z-WORLD, 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 Z-WORLD'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 Z-WORLD. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Z-WORLD.

Copyright 2004 Z-World, Inc. All rights reserved.