# Table of Contents

- Alphabetical Listing of Dynamic C Functions  
  v

- Group Listing of Dynamic C Functions  
  xv

- Chapter 1: Function Descriptions  
  1

- Software License Agreement  
  557
Alphabetical Listing of Dynamic C Functions

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

Symbols

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_GetSysMacroIndex</td>
<td>178</td>
</tr>
<tr>
<td>_GetSysMacroValue</td>
<td>179</td>
</tr>
<tr>
<td>_sysIsSoftReset</td>
<td>517</td>
</tr>
<tr>
<td>xalloc</td>
<td>542</td>
</tr>
<tr>
<td>xavail</td>
<td>545</td>
</tr>
<tr>
<td>abs</td>
<td>2</td>
</tr>
<tr>
<td>acos</td>
<td>2</td>
</tr>
<tr>
<td>acot</td>
<td>3</td>
</tr>
<tr>
<td>asec</td>
<td>3</td>
</tr>
<tr>
<td>AESdecrypt4x4</td>
<td>4</td>
</tr>
<tr>
<td>AESdecryptStream4x4_CBC</td>
<td>5</td>
</tr>
<tr>
<td>AESencrypt4x4</td>
<td>6</td>
</tr>
<tr>
<td>AESencryptStream4x4_CBC</td>
<td>7</td>
</tr>
<tr>
<td>AESexpandKey4</td>
<td>8</td>
</tr>
<tr>
<td>AESinitStream4x4</td>
<td>9</td>
</tr>
<tr>
<td>asec</td>
<td>10</td>
</tr>
<tr>
<td>asin</td>
<td>10</td>
</tr>
<tr>
<td>atan</td>
<td>11</td>
</tr>
<tr>
<td>atan2</td>
<td>12</td>
</tr>
<tr>
<td>atof</td>
<td>13</td>
</tr>
<tr>
<td>atoi</td>
<td>14</td>
</tr>
<tr>
<td>atol</td>
<td>15</td>
</tr>
<tr>
<td>BIT</td>
<td>17</td>
</tr>
<tr>
<td>bit</td>
<td>16</td>
</tr>
<tr>
<td>BitRdPortE</td>
<td>18</td>
</tr>
<tr>
<td>BitRdPortI</td>
<td>19</td>
</tr>
<tr>
<td>BitWrPortE</td>
<td>20</td>
</tr>
<tr>
<td>BitWrPortI</td>
<td>21</td>
</tr>
<tr>
<td>CalculateECC256</td>
<td>22</td>
</tr>
<tr>
<td>ceil</td>
<td>24</td>
</tr>
<tr>
<td>ChkCorrectECC256</td>
<td>23</td>
</tr>
<tr>
<td>chkHardReset</td>
<td>25</td>
</tr>
<tr>
<td>chkSoftReset</td>
<td>25</td>
</tr>
<tr>
<td>chkWDTO</td>
<td>26</td>
</tr>
<tr>
<td>clockDoublerOff</td>
<td>27</td>
</tr>
<tr>
<td>clockDoublerOn</td>
<td>27</td>
</tr>
<tr>
<td>CloseInputCompressedFile</td>
<td>28</td>
</tr>
<tr>
<td>CloseOutputCompressedFile</td>
<td>28</td>
</tr>
<tr>
<td>CoBegin</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktAreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktAsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktBreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktBsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktCreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktCsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktDreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktDsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktEreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktEsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktFsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktGgets</td>
<td>31</td>
</tr>
<tr>
<td>cof_pktGgets</td>
<td>32</td>
</tr>
<tr>
<td>cof_pktAputec</td>
<td>33</td>
</tr>
<tr>
<td>cof_pktAputes</td>
<td>34</td>
</tr>
<tr>
<td>cof_pktAread</td>
<td>35</td>
</tr>
<tr>
<td>cof_pktAwrites</td>
<td>36</td>
</tr>
<tr>
<td>cof_pktBgetc</td>
<td>31</td>
</tr>
<tr>
<td>cof_pktBgets</td>
<td>32</td>
</tr>
<tr>
<td>cof_pktBputc</td>
<td>33</td>
</tr>
<tr>
<td>cof_pktBputs</td>
<td>34</td>
</tr>
<tr>
<td>cof_pktBread</td>
<td>35</td>
</tr>
<tr>
<td>cof_pktBwrite</td>
<td>36</td>
</tr>
<tr>
<td>cof_pktCgetc</td>
<td>31</td>
</tr>
<tr>
<td>cof_pktCgets</td>
<td>32</td>
</tr>
<tr>
<td>cof_pktCputc</td>
<td>33</td>
</tr>
<tr>
<td>cof_pktCputs</td>
<td>34</td>
</tr>
<tr>
<td>cof_pktCread</td>
<td>35</td>
</tr>
<tr>
<td>cof_pktCwrite</td>
<td>36</td>
</tr>
<tr>
<td>cof_pktDgetc</td>
<td>31</td>
</tr>
<tr>
<td>cof_pktDgets</td>
<td>32</td>
</tr>
<tr>
<td>cof_pktDputc</td>
<td>33</td>
</tr>
<tr>
<td>cof_pktDputs</td>
<td>34</td>
</tr>
</tbody>
</table>
cof_serDread .................................................35
cof_serDwrite .................................................36
cof_serEgete ..................................................31
cof_serEgets ...................................................32
cof_serEpute ...................................................33
cof_serEputs ...................................................34
cof_serEread ...................................................35
cof_serEwrite ...................................................36
cof_serFgete ...................................................31
cof_serFgets ...................................................32
cof_serFpute ...................................................33
cof_serFputs ...................................................34
cs ...............................................................40
cosh .............................................................40

D
DecompressFile .............................................41
defineErrorHandler .........................................42
deg .............................................................43
DelayMs .......................................................43
DelaySec .......................................................44
DelayTicks .....................................................45
Disable_HW_WDT ..........................................45
DMAalloc ......................................................47
DMAcompleted ...............................................48
DMAhandle2chan .........................................49
DMAioe2mem ..................................................50
DMAioi2mem ..................................................52
DMAloadBufDesc ..........................................53
DMAmatchSetup ............................................54
DMAmem2iec ..................................................55
DMAmem2ioi ..................................................56
DMAmem2mem ..................................................57
DMApoll .......................................................58
DMAprintBufDesc .........................................59
DMAprintReg ...............................................60
DMAsetBufDesc .............................................61
DMAsetDirect ...............................................62
DMAsetParameters ........................................63
DMAstartAuto ..............................................64
DMAstartDirect ..........................................65
DMAstop ........................................................66
DMAstopDirect .............................................67
DMAtimerSetup ............................................67
DMAunalloc ..................................................68
enableIObus ..................................................69
erlogFormatEntry ...........................................71
erlogFormatRegDump .......................................72
erlogFormatStackDump .....................................72
erlogGetHeaderInfo .......................................70
erlogGetMessage ...........................................73
erlogGetNthEntry ..........................................71
erlogReadHeader ..........................................73
error_message .............................................74
exception ....................................................75
exp .............................................................76

F
fabs ............................................................77
fat_AutoMount ............................................78
fat_Close .....................................................81
fat_CreateDir ...............................................82
fat_CreateFile .............................................83
fat_CreateTime ............................................84
fat_Delete ....................................................85
fat_NumDevice .............................................86
fat_NumPartition ..........................................87
fat_FileSize ................................................88
fat_FormatDevice .........................................89
fat_FormatPartition ......................................90
fat_Free ......................................................91
fat_GetAttr ..................................................92
fatGetName ..................................................93
fat_GetPartition ..........................................94
fat_Init ......................................................95
fat_InitUCOSMutex ........................................96
fat_IsClosed ...............................................97
fat_IsOpen ..................................................98
fat_LastAccess ............................................99
fat_LastWrite .............................................100
fat_MousePartition ......................................101
fat_Open ......................................................102
fat_OpenDir ...............................................104
fat_PartitionDevice .....................................105
fat_Read .....................................................106
fat_ReadDir .................................................107
fat SEEK ......................................................109
fat_SetAttr ...............................................111
fat_Split ....................................................112
fat_Status ...................................................113
fat_SyncFile ..............................................114
fat_SyncPartition ........................................115
fat_Tell ......................................................116
fat_tick .......................................................117
fat_Truncate ..............................................118
fat_UnmountDevice ....................................119
fat_UnMountPartition ................................120
fat_Write ....................................................121
fat_xRead ..................................................122, 123
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic C Functions</td>
<td></td>
</tr>
<tr>
<td>fwrite (FS2)</td>
<td>174</td>
</tr>
<tr>
<td>fwrite (FS1)</td>
<td>173</td>
</tr>
<tr>
<td>ftell (FS2)</td>
<td>170</td>
</tr>
<tr>
<td>ftell (FS1)</td>
<td>169</td>
</tr>
<tr>
<td>fshift</td>
<td>171</td>
</tr>
<tr>
<td>fseek (FS2)</td>
<td>159</td>
</tr>
<tr>
<td>fseek (FS1)</td>
<td>158</td>
</tr>
<tr>
<td>fsck</td>
<td>157</td>
</tr>
<tr>
<td>fs_sync (FS2)</td>
<td>168</td>
</tr>
<tr>
<td>fs_set_lx (FS2)</td>
<td>165</td>
</tr>
<tr>
<td>fs_setup (FS2)</td>
<td>166</td>
</tr>
<tr>
<td>fs_close</td>
<td>125</td>
</tr>
<tr>
<td>fs_create</td>
<td>126</td>
</tr>
<tr>
<td>fs_create (FS2)</td>
<td>127</td>
</tr>
<tr>
<td>fs_create (FS1)</td>
<td>128</td>
</tr>
<tr>
<td>fs_create_unused</td>
<td>129</td>
</tr>
<tr>
<td>fs_create_unused (FS2)</td>
<td>130</td>
</tr>
<tr>
<td>fs_delete</td>
<td>131</td>
</tr>
<tr>
<td>fs_delete (FS2)</td>
<td>132</td>
</tr>
<tr>
<td>fs_delete (FS1)</td>
<td>133</td>
</tr>
<tr>
<td>fsFlush (FS2)</td>
<td>134</td>
</tr>
<tr>
<td>fsFlush (FS1)</td>
<td>135</td>
</tr>
<tr>
<td>fs_get_lx_size (FS2)</td>
<td>136</td>
</tr>
<tr>
<td>fs_get_lx (FS2)</td>
<td>137</td>
</tr>
<tr>
<td>fs_get_lx (FS1)</td>
<td>138</td>
</tr>
<tr>
<td>fs_get_other_lx (FS2)</td>
<td>139</td>
</tr>
<tr>
<td>fs_get_other_lx (FS1)</td>
<td>140</td>
</tr>
<tr>
<td>fs_get_ram_lx (FS2)</td>
<td>141</td>
</tr>
<tr>
<td>fs_get_ram_lx (FS1)</td>
<td>142</td>
</tr>
<tr>
<td>fs_init (FS1)</td>
<td>143</td>
</tr>
<tr>
<td>fs_init (FS2)</td>
<td>144</td>
</tr>
<tr>
<td>fs_format (FS2)</td>
<td>145</td>
</tr>
<tr>
<td>fs_format (FS1)</td>
<td>146</td>
</tr>
<tr>
<td>fs_format (FS2)</td>
<td>147</td>
</tr>
<tr>
<td>fsformatl (FS2)</td>
<td>148</td>
</tr>
<tr>
<td>fsformatl (FS1)</td>
<td>149</td>
</tr>
<tr>
<td>fsformatl (FS2)</td>
<td>150</td>
</tr>
<tr>
<td>fsformatl (FS1)</td>
<td>151</td>
</tr>
<tr>
<td>fsdelete (FS2)</td>
<td>152</td>
</tr>
<tr>
<td>fsdelete (FS1)</td>
<td>153</td>
</tr>
<tr>
<td>fscreate (FS2)</td>
<td>154</td>
</tr>
<tr>
<td>fscreate (FS1)</td>
<td>155</td>
</tr>
<tr>
<td>fscreate (FS2)</td>
<td>156</td>
</tr>
<tr>
<td>fsreserve Bloclks (FS2)</td>
<td>157</td>
</tr>
<tr>
<td>fsreserve Bloclks (FS1)</td>
<td>158</td>
</tr>
<tr>
<td>get_cpu_frequency</td>
<td>176</td>
</tr>
<tr>
<td>get_long</td>
<td>177</td>
</tr>
<tr>
<td>getchar</td>
<td>178</td>
</tr>
<tr>
<td>getere</td>
<td>179</td>
</tr>
<tr>
<td>getdivider19200</td>
<td>180</td>
</tr>
<tr>
<td>gets</td>
<td>181</td>
</tr>
<tr>
<td>GetVectExtern2000</td>
<td>182</td>
</tr>
<tr>
<td>GetVectExtern3000</td>
<td>183</td>
</tr>
<tr>
<td>GetVectInter</td>
<td>184</td>
</tr>
<tr>
<td>gps_get_position</td>
<td>185</td>
</tr>
<tr>
<td>gps_get_utc</td>
<td>186</td>
</tr>
<tr>
<td>gps_ground_distance</td>
<td>187</td>
</tr>
<tr>
<td>hanncplx</td>
<td>188</td>
</tr>
<tr>
<td>hamreal</td>
<td>189</td>
</tr>
<tr>
<td>HDLCabortE</td>
<td>190</td>
</tr>
<tr>
<td>HDLCabortF</td>
<td>191</td>
</tr>
<tr>
<td>HDLCcloseE</td>
<td>192</td>
</tr>
<tr>
<td>HDLCcloseF</td>
<td>193</td>
</tr>
<tr>
<td>HDLCdropE</td>
<td>194</td>
</tr>
<tr>
<td>HDLCdropF</td>
<td>195</td>
</tr>
<tr>
<td>HDLCerrorE</td>
<td>196</td>
</tr>
<tr>
<td>HDLCerrorF</td>
<td>197</td>
</tr>
<tr>
<td>HDLCextClockE</td>
<td>198</td>
</tr>
<tr>
<td>HDLCextClockF</td>
<td>199</td>
</tr>
<tr>
<td>HDLCopenE</td>
<td>200</td>
</tr>
<tr>
<td>HDLCopenF</td>
<td>201</td>
</tr>
<tr>
<td>HDLCpeekE</td>
<td>202</td>
</tr>
<tr>
<td>HDLCpeekF</td>
<td>203</td>
</tr>
<tr>
<td>HDLCreceiveE</td>
<td>204</td>
</tr>
<tr>
<td>HDLCreceiveF</td>
<td>205</td>
</tr>
<tr>
<td>HDLCsendE</td>
<td>206</td>
</tr>
<tr>
<td>HDLCsendF</td>
<td>207</td>
</tr>
<tr>
<td>HDLCsendingE</td>
<td>208</td>
</tr>
<tr>
<td>HDLCsendingF</td>
<td>209</td>
</tr>
<tr>
<td>hexstrtobyte</td>
<td>210</td>
</tr>
<tr>
<td>hitwd</td>
<td>211</td>
</tr>
<tr>
<td>htoa</td>
<td>212</td>
</tr>
<tr>
<td>IntervalMs</td>
<td>213</td>
</tr>
<tr>
<td>IntervalSec</td>
<td>214</td>
</tr>
<tr>
<td>IntervalTick</td>
<td>215</td>
</tr>
<tr>
<td>i2c_check_ack</td>
<td>206</td>
</tr>
<tr>
<td>i2c_init</td>
<td>207</td>
</tr>
<tr>
<td>i2c_read_char</td>
<td>208</td>
</tr>
<tr>
<td>i2c_send_ack</td>
<td>209</td>
</tr>
<tr>
<td>i2c_send_nak</td>
<td>210</td>
</tr>
<tr>
<td>i2c_start_tx</td>
<td>211</td>
</tr>
<tr>
<td>i2c_startw_tx</td>
<td>212</td>
</tr>
<tr>
<td>i2c_stop_tx</td>
<td>213</td>
</tr>
<tr>
<td>i2c_write_char</td>
<td>214</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>labs</td>
<td>212</td>
</tr>
<tr>
<td>ldexp</td>
<td>213</td>
</tr>
<tr>
<td>log</td>
<td>213</td>
</tr>
<tr>
<td>log_clean</td>
<td>214</td>
</tr>
<tr>
<td>log_close</td>
<td>215</td>
</tr>
<tr>
<td>log_condition</td>
<td>216</td>
</tr>
<tr>
<td>log_format</td>
<td>217</td>
</tr>
<tr>
<td>log_map</td>
<td>218</td>
</tr>
<tr>
<td>log_next</td>
<td>219</td>
</tr>
<tr>
<td>log_open</td>
<td>220</td>
</tr>
<tr>
<td>log_prev</td>
<td>221</td>
</tr>
<tr>
<td>log_put</td>
<td>222</td>
</tr>
<tr>
<td>log_seek</td>
<td>223</td>
</tr>
<tr>
<td>log10</td>
<td>224</td>
</tr>
<tr>
<td>longjmp</td>
<td>224</td>
</tr>
<tr>
<td>loophead</td>
<td>225</td>
</tr>
<tr>
<td>loopinit</td>
<td>225</td>
</tr>
<tr>
<td>lsqrt</td>
<td>226</td>
</tr>
<tr>
<td>ltoa</td>
<td>226</td>
</tr>
<tr>
<td>ltoan</td>
<td>227</td>
</tr>
<tr>
<td>lx_format</td>
<td>228</td>
</tr>
<tr>
<td>mbr_CreatePartition</td>
<td>229</td>
</tr>
<tr>
<td>mbr_EnumDevice</td>
<td>230</td>
</tr>
<tr>
<td>mbr_FormatDevice</td>
<td>231</td>
</tr>
<tr>
<td>mbr_MountPartition</td>
<td>232</td>
</tr>
<tr>
<td>mbr_UnmountPartition</td>
<td>233</td>
</tr>
<tr>
<td>mbr.ValidatePartitions</td>
<td>234</td>
</tr>
<tr>
<td>md5_append</td>
<td>235</td>
</tr>
<tr>
<td>md5_finish</td>
<td>236</td>
</tr>
<tr>
<td>md5_ini</td>
<td>235</td>
</tr>
<tr>
<td>memchr</td>
<td>237</td>
</tr>
<tr>
<td>memcmp</td>
<td>238</td>
</tr>
<tr>
<td>memcpy</td>
<td>239</td>
</tr>
<tr>
<td>memmove</td>
<td>240</td>
</tr>
<tr>
<td>memset</td>
<td>241</td>
</tr>
<tr>
<td>mktime</td>
<td>242</td>
</tr>
<tr>
<td>mtkm</td>
<td>243</td>
</tr>
<tr>
<td>modf</td>
<td>244</td>
</tr>
<tr>
<td>nf_eraseBlock</td>
<td>245</td>
</tr>
<tr>
<td>nf_getPageCount</td>
<td>246</td>
</tr>
<tr>
<td>nf_getPageSize</td>
<td>247</td>
</tr>
<tr>
<td>nf_initDevice</td>
<td>248</td>
</tr>
<tr>
<td>nf_InitDriver</td>
<td>250</td>
</tr>
<tr>
<td>nf_isBusyRBHW</td>
<td>251</td>
</tr>
<tr>
<td>nf_isBusyStatus</td>
<td>252</td>
</tr>
<tr>
<td>nf_readPage</td>
<td>253</td>
</tr>
<tr>
<td>nf_writePage</td>
<td>254</td>
</tr>
<tr>
<td>nf_XD_Detect</td>
<td>255</td>
</tr>
<tr>
<td>OpenInputCompressedFile</td>
<td>256</td>
</tr>
<tr>
<td>OpenOutputCompressedFile</td>
<td>257</td>
</tr>
<tr>
<td>OS_ENTER_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OS_EXIT_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OSFlagAccept</td>
<td>259</td>
</tr>
<tr>
<td>OSFlagCreate</td>
<td>261</td>
</tr>
<tr>
<td>OSFlagDel</td>
<td>262</td>
</tr>
<tr>
<td>OSFlagPend</td>
<td>263</td>
</tr>
<tr>
<td>OSFlagPost</td>
<td>265</td>
</tr>
<tr>
<td>OSFlagQuery</td>
<td>266</td>
</tr>
<tr>
<td>OSInit</td>
<td>267</td>
</tr>
<tr>
<td>OSMboxAccept</td>
<td>267</td>
</tr>
<tr>
<td>OSMboxCreate</td>
<td>268</td>
</tr>
<tr>
<td>OSMboxDel</td>
<td>269</td>
</tr>
<tr>
<td>OSMboxPend</td>
<td>270</td>
</tr>
<tr>
<td>OSMboxPost</td>
<td>271</td>
</tr>
<tr>
<td>OSMboxPostOpt</td>
<td>272</td>
</tr>
<tr>
<td>OSMboxQuery</td>
<td>273</td>
</tr>
<tr>
<td>OSMemCreate</td>
<td>274</td>
</tr>
<tr>
<td>OSMemGet</td>
<td>275</td>
</tr>
<tr>
<td>OSMemPut</td>
<td>276</td>
</tr>
<tr>
<td>OSMemQuery</td>
<td>277</td>
</tr>
<tr>
<td>OSMutexAccept</td>
<td>278</td>
</tr>
<tr>
<td>OSMutexCreate</td>
<td>279</td>
</tr>
<tr>
<td>OSMutexDel</td>
<td>280</td>
</tr>
<tr>
<td>OSMutexPend</td>
<td>281</td>
</tr>
<tr>
<td>OSMutexPost</td>
<td>282</td>
</tr>
<tr>
<td>OSMutexQuery</td>
<td>283</td>
</tr>
<tr>
<td>OSQAccept</td>
<td>284</td>
</tr>
<tr>
<td>OSQCreate</td>
<td>285</td>
</tr>
<tr>
<td>OSQDel</td>
<td>286</td>
</tr>
<tr>
<td>OSQFlush</td>
<td>287</td>
</tr>
</tbody>
</table>
pfree .......................................................... 329
pfirst .......................................................... 327
pcalloc ....................................................... 326
pavail_fast .................................................. 325
palloc ......................................................... 324
paddrSS ..................................................... 321
paddrDS ..................................................... 320
paddr ......................................................... 319
outchrs ....................................................... 317
outstr ......................................................... 318
phwm .......................................................... 331
pktAclose .................................................... 332
pktAgetErrors .............................................. 332
pktAinitBuffers ........................................... 333
pktAopen .................................................... 334
pktAreceive .................................................. 336
pktAsend ...................................................... 337
pktBsend ...................................................... 337
pktBsetParity .............................................. 338
pktCclose ..................................................... 332
pktCgetErrors .............................................. 332
pktCinitBuffers ........................................... 333
pktCopen ..................................................... 334
pktCreceive .................................................. 336
pktCsend ...................................................... 337
pktCsetParity .............................................. 338
pktDclose ..................................................... 332
pktDgetErrors .............................................. 332
pktDinitBuffers ........................................... 333
pktDopen ..................................................... 334
pktDreceive .................................................. 336
pktDsend ...................................................... 337
pktDsetParity .............................................. 338
pktEcclose .................................................. 332
pktEgetErrors .............................................. 332
pktEinitBuffers ........................................... 333
pktEopen ..................................................... 334
pktEreceive .................................................. 336
pktEsend ...................................................... 337
pktEsetParity .............................................. 338
pktFclose ..................................................... 332
pktFgetErrors .............................................. 332
pktFinitBuffers ........................................... 333
pktFopen ..................................................... 334
pktFreceive .................................................. 336
pktFsend ...................................................... 337
pktFsending .................................................. 338
pktFsetParity .............................................. 338
plast .......................................................... 339
plast .......................................................... 340
pmovebetween .............................................. 341
pmovebetween_fast ....................................... 343
readUserBlock ..................................................393
readUserBlockArray ...........................................394
registry_enumerate ...........................................395
registry_finish_read ...........................................398
registry_finish_write .........................................399
registry_get ..................................................397
registry_prep_read ...........................................400
registry_prep_write ..........................................403
registry_read ..................................................405
registry_update ................................................406
registry_write ..................................................407
 RES ......................................................................411
res .......................................................................411
ResetErrorLog ....................................................412
root2xmem .........................................................413
device .................................................................414
runwatch ............................................................415

S
sdspi_debounce ..................................................415
sdspi_get_csd .....................................................416
sdspi_get_scr .....................................................417
sdspi_get_status_reg ............................................418
sdspi_getSectorCount ..........................................418
sdspi_init_card ..................................................419
sdspi_initDevice ...............................................420
sdspi_isWriting ..................................................421
sdspi_notbusy ....................................................421
sdspi_print_dev ..................................................422
sdspi_process_command .........................................423
sdspi_read_sector .................................................424
sdspi_reset_card ..................................................425
sdspi_sendingAP ..................................................426
sdspi_set_block_length ........................................427
sdspi_setLED .....................................................426
sdspi_write_sector ..............................................429
sdspi_WriteContinue ..........................................428
serAclose .........................................................448
serAclose .........................................................448
serAmaOff .........................................................449
serAmaOn ..........................................................450
serAflowcontrolOff .............................................451
serAflowcontrolOn ..............................................452
serAgetc ............................................................453
serAgetError ......................................................454
serAopen ...........................................................455
serAparity ..........................................................456
serApeek ............................................................457
serAputc ............................................................458
serAputs ............................................................459
serArdFlush .........................................................460
serArdFree ..........................................................460
serArdUsed ..........................................................461
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>useClockDivider</td>
<td>525</td>
</tr>
<tr>
<td>useClockDivider3000</td>
<td>526</td>
</tr>
<tr>
<td>useMainOsc</td>
<td>527</td>
</tr>
<tr>
<td>utoa</td>
<td>527</td>
</tr>
<tr>
<td><strong>V</strong></td>
<td></td>
</tr>
<tr>
<td>VdGetFreeWd</td>
<td>529</td>
</tr>
<tr>
<td>VdInit</td>
<td>530</td>
</tr>
<tr>
<td>VdReleaseWd</td>
<td>531</td>
</tr>
<tr>
<td>vram2root</td>
<td>528</td>
</tr>
<tr>
<td><strong>W</strong></td>
<td></td>
</tr>
<tr>
<td>write_rtc</td>
<td>534</td>
</tr>
<tr>
<td>WriteFlash2</td>
<td>532</td>
</tr>
<tr>
<td>WriteFlash2Array</td>
<td>533</td>
</tr>
<tr>
<td>writeUserBlock</td>
<td>535</td>
</tr>
<tr>
<td>writeUserBlockArray</td>
<td>537</td>
</tr>
<tr>
<td>WrPortE</td>
<td>539</td>
</tr>
<tr>
<td>WrPortI</td>
<td>540</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td></td>
</tr>
<tr>
<td>xalloc</td>
<td>541</td>
</tr>
<tr>
<td>xalloc_stats</td>
<td>543</td>
</tr>
<tr>
<td>xavail</td>
<td>544</td>
</tr>
<tr>
<td>xCalculateECC256</td>
<td>546</td>
</tr>
<tr>
<td>xChkCorrectECC256</td>
<td>547</td>
</tr>
<tr>
<td>xgetfloat</td>
<td>548</td>
</tr>
<tr>
<td>xgetint</td>
<td>548</td>
</tr>
<tr>
<td>xgetlong</td>
<td>549</td>
</tr>
<tr>
<td>xmem2root</td>
<td>550</td>
</tr>
<tr>
<td>xmem2xmem</td>
<td>551</td>
</tr>
<tr>
<td>xmemchr</td>
<td>552</td>
</tr>
<tr>
<td>xmemcmp</td>
<td>553</td>
</tr>
<tr>
<td>xrelease</td>
<td>554</td>
</tr>
<tr>
<td>xsetfloat</td>
<td>555</td>
</tr>
<tr>
<td>xsetint</td>
<td>555</td>
</tr>
<tr>
<td>xsetlong</td>
<td>556</td>
</tr>
<tr>
<td>xstrlen</td>
<td>556</td>
</tr>
</tbody>
</table>
## Group Listing of Dynamic C Functions

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

### A

**Arithmetic**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>2</td>
</tr>
<tr>
<td>getc</td>
<td>176</td>
</tr>
<tr>
<td>lsqrt</td>
<td>226</td>
</tr>
</tbody>
</table>

### B

**Bit Manipulation**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>17</td>
</tr>
<tr>
<td>bit</td>
<td>16</td>
</tr>
<tr>
<td>RES</td>
<td>411</td>
</tr>
<tr>
<td>res</td>
<td>411</td>
</tr>
<tr>
<td>SET</td>
<td>466</td>
</tr>
<tr>
<td>set</td>
<td>466</td>
</tr>
</tbody>
</table>

### Bus Operation (Rabbit 3000, 4000)

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>disableIObus</td>
<td>46</td>
</tr>
<tr>
<td>enableIObus</td>
<td>69</td>
</tr>
</tbody>
</table>

### C

**Character**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum</td>
<td>197</td>
</tr>
<tr>
<td>isalpha</td>
<td>198</td>
</tr>
<tr>
<td>iscntrl</td>
<td>198</td>
</tr>
<tr>
<td>isdigit</td>
<td>200</td>
</tr>
<tr>
<td>isgraph</td>
<td>201</td>
</tr>
<tr>
<td>islower</td>
<td>201</td>
</tr>
<tr>
<td>isprint</td>
<td>202</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispunct</td>
<td>203</td>
</tr>
<tr>
<td>isspace</td>
<td>202</td>
</tr>
<tr>
<td>isupper</td>
<td>204</td>
</tr>
<tr>
<td>isxdigit</td>
<td>204</td>
</tr>
</tbody>
</table>

### D

**Data Encryption**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESdecrypt4x4</td>
<td>4</td>
</tr>
<tr>
<td>AESdecryptStream4x4_CBC</td>
<td>5</td>
</tr>
<tr>
<td>AESencrypt4x4</td>
<td>6</td>
</tr>
<tr>
<td>AESencryptStream4x4_CBC</td>
<td>7</td>
</tr>
<tr>
<td>AESexpandKey4</td>
<td>8</td>
</tr>
<tr>
<td>AESinitStream4x4</td>
<td>9</td>
</tr>
</tbody>
</table>

**Direct Memory Access (Rabbit 4000, 5000)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAalloc</td>
<td>47</td>
</tr>
<tr>
<td>DMAcompleted</td>
<td>48</td>
</tr>
<tr>
<td>DMAhandle2chan</td>
<td>49</td>
</tr>
<tr>
<td>DMAioe2mem</td>
<td>50</td>
</tr>
<tr>
<td>DMAioi2mem</td>
<td>52</td>
</tr>
<tr>
<td>DMAloadBufDesc</td>
<td>53</td>
</tr>
<tr>
<td>DMAmatchSetup</td>
<td>54</td>
</tr>
<tr>
<td>DMAmem2ioe</td>
<td>55</td>
</tr>
<tr>
<td>DMAmem2ioi</td>
<td>56</td>
</tr>
<tr>
<td>DMAmem2mem</td>
<td>57</td>
</tr>
<tr>
<td>DMAPoll</td>
<td>58</td>
</tr>
<tr>
<td>DMAPrintBufDesc</td>
<td>59</td>
</tr>
<tr>
<td>DMAPrintRegs</td>
<td>60</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>errlogGetMessage</td>
<td>73</td>
</tr>
<tr>
<td>errlogGetNthEntry</td>
<td>71</td>
</tr>
<tr>
<td>errlogReadHeader</td>
<td>73</td>
</tr>
<tr>
<td>error_message</td>
<td>74</td>
</tr>
<tr>
<td>exception</td>
<td>75</td>
</tr>
<tr>
<td>ResetErrorLog</td>
<td>412</td>
</tr>
<tr>
<td><strong>Extended Memory</strong></td>
<td></td>
</tr>
<tr>
<td>_xalloc</td>
<td>542</td>
</tr>
<tr>
<td>_xavail</td>
<td>545</td>
</tr>
<tr>
<td>paddr</td>
<td>319</td>
</tr>
<tr>
<td>paddrDS</td>
<td>320</td>
</tr>
<tr>
<td>paddrSS</td>
<td>321</td>
</tr>
<tr>
<td>root2xmem</td>
<td>413</td>
</tr>
<tr>
<td>xalloc</td>
<td>541</td>
</tr>
<tr>
<td>xalloc_stats</td>
<td>543</td>
</tr>
<tr>
<td>xavail</td>
<td>544</td>
</tr>
<tr>
<td>xgetfloat</td>
<td>548</td>
</tr>
<tr>
<td>xgetint</td>
<td>548</td>
</tr>
<tr>
<td>xgetlong</td>
<td>549</td>
</tr>
<tr>
<td>xmem2root</td>
<td>550</td>
</tr>
<tr>
<td>xmem2xmem</td>
<td>551</td>
</tr>
<tr>
<td>xmemchr</td>
<td>552</td>
</tr>
<tr>
<td>xmemcmp</td>
<td>553</td>
</tr>
<tr>
<td>xrelease</td>
<td>554</td>
</tr>
<tr>
<td>xsetfloat</td>
<td>555</td>
</tr>
<tr>
<td>xsetint</td>
<td>555</td>
</tr>
<tr>
<td>xsetlong</td>
<td>556</td>
</tr>
<tr>
<td>xstrlen</td>
<td>556</td>
</tr>
<tr>
<td><strong>File Compression</strong></td>
<td></td>
</tr>
<tr>
<td>CloseInputCompressedFile</td>
<td>28</td>
</tr>
<tr>
<td>CloseOutputCompressedFile</td>
<td>28</td>
</tr>
<tr>
<td>CompressFile</td>
<td>37</td>
</tr>
<tr>
<td>DecompressFile</td>
<td>41</td>
</tr>
<tr>
<td>OpenInputCompressedFile</td>
<td>256</td>
</tr>
<tr>
<td>OpenOutputCompressedFile</td>
<td>257</td>
</tr>
<tr>
<td>ReadCompressedFile</td>
<td>391</td>
</tr>
<tr>
<td><strong>File System, FAT</strong></td>
<td></td>
</tr>
<tr>
<td>fat_AutoMount</td>
<td>78</td>
</tr>
<tr>
<td>fat_Close</td>
<td>81</td>
</tr>
<tr>
<td>fat_CreateDir</td>
<td>82</td>
</tr>
<tr>
<td>fat_CreateFile</td>
<td>83</td>
</tr>
<tr>
<td>fat_CreateTime</td>
<td>84</td>
</tr>
<tr>
<td>fat_Delete</td>
<td>85</td>
</tr>
<tr>
<td>fat_NumDevice</td>
<td>86</td>
</tr>
<tr>
<td>fat_NumPartition</td>
<td>87</td>
</tr>
<tr>
<td>fat_FileSize</td>
<td>88</td>
</tr>
<tr>
<td>fat_FormatDevice</td>
<td>89</td>
</tr>
<tr>
<td>fat_FormatPartition</td>
<td>90</td>
</tr>
<tr>
<td>fat_Free</td>
<td>91</td>
</tr>
<tr>
<td>fat_GetAttr</td>
<td>92</td>
</tr>
<tr>
<td>fat_GetName</td>
<td>93</td>
</tr>
<tr>
<td>fat_GetPartition</td>
<td>94</td>
</tr>
<tr>
<td>fat_Init</td>
<td>95</td>
</tr>
<tr>
<td>fat_InitUCOSMutex</td>
<td>96</td>
</tr>
<tr>
<td>fat_LockedDevice</td>
<td>97</td>
</tr>
<tr>
<td>fat_IsClosed</td>
<td>98</td>
</tr>
<tr>
<td>fat_LastAccess</td>
<td>99</td>
</tr>
<tr>
<td>fat_LastWrite</td>
<td>100</td>
</tr>
<tr>
<td>fat_MountPartition</td>
<td>101</td>
</tr>
<tr>
<td>fat_Open</td>
<td>102</td>
</tr>
<tr>
<td>fat_OpenDir</td>
<td>104</td>
</tr>
<tr>
<td>fat_PartitionDevice</td>
<td>105</td>
</tr>
<tr>
<td>fat_Read</td>
<td>106</td>
</tr>
<tr>
<td>fat_ReadDir</td>
<td>107</td>
</tr>
<tr>
<td>fat_Write</td>
<td>111</td>
</tr>
</tbody>
</table>

**Fast Fourier Transforms**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fftcplx</td>
<td>133</td>
</tr>
<tr>
<td>fftcplxinv</td>
<td>134</td>
</tr>
<tr>
<td>fitreal</td>
<td>135</td>
</tr>
<tr>
<td>fitrealinv</td>
<td>136</td>
</tr>
<tr>
<td>hancplx</td>
<td>184</td>
</tr>
<tr>
<td>hanreal</td>
<td>185</td>
</tr>
<tr>
<td>powerspectrum</td>
<td>354</td>
</tr>
</tbody>
</table>
fat_Split ..............................................112
fat_Status ............................................113
fat_SyncFile ........................................114
fat_SyncPartition .................................115
fat_Tell ...............................................116
fat_tick ...............................................117
fat_Truncate ........................................118
fat_UnmountDevice .............................119
fat_UnmountPartition ..........................120
fat_Write ............................................121
fat_xRead ...................................122, 123
fat_xWrite ...........................................124

File System, FS1
  fcreate ..........................................126
  fcreate_unused ..................................128
  fdelete ..........................................130
  fopen_rd ........................................146
  fopen_wr ........................................148
  fread .............................................150
  fs_format .......................................153
  fs_init .........................................155
  fs_reserve_blocks .........................157
  fsck ............................................157
  fseek ............................................158
  ftell .............................................169
  fwrite .........................................173

File System, FS2
  fclose ..........................................125
  fcreate ..........................................127
  fcreate_unused ................................129
  fdelete ..........................................131
  fflush ..........................................132
  fopen_rd ........................................147
  fopen_wr ........................................149
  fread .............................................151
  fs_format .......................................154
  fs_get_flash_lx .............................160
  fs_get_lx .......................................161
  fs_get_lx_size ................................162
  fs_get_other_lx ................................163
  fs_get_ram_lx ................................164
  fs_init .........................................156
  fs_set_lx .......................................165
  fs_setup ........................................166
  fs_sync .........................................168
  fseek ..........................................159
  fshift ..........................................171
  ftell .............................................170
  fwrite .........................................174
  lx_format ......................................228

File System, Registry
  registry_enumerate ............................395
  registry_finish_read ..........................398
  registry_finish_write ........................399
  registry_get ....................................397
  registry_prep_read ............................400
  registry_prep_write ...........................403
  registry_read ...................................405
  registry_update ...............................406
  registry_write ..................................407

Flash, NAND
  nf_eraseBlock ..................................245
  nf_getPageCount ................................246
  nf_getPageSize ............................247
  nf_initDevice ..................................248
  nf_InitDriver ..................................250
  nf_isBusyRBHW ..................................251
  nf_isBusyStatus ................................252
  nf_readPage ....................................253
  nf_writePage ...................................254
  nf_XD_Detect ...................................255

Flash, Parallel
  flash_erasechip ..................................137
  flash_erasesector ................................138
flash_gettype .......................................139
flash_init .............................................140
flash_read ...........................................141
flash_readsector ..................................142
flash_sector2xwindow .........................143
flash_writesector .................................144
WriteFlash2 ........................................532
WriteFlash2Array ................................533

Floating-Point Math
acos ........................................................2
acot ........................................................3
acsc ........................................................3
asec ......................................................10
asin .......................................................10
atan ......................................................11
atan2 ....................................................12
ceil .......................................................24
cos .......................................................40
cosh ......................................................40
deg .......................................................43
exp .......................................................76
fabs ......................................................77
floor ......................................................145
fmod ....................................................145
frexp ...................................................152
labs .....................................................212
ldexp ..................................................213
log ......................................................213
log10 ..................................................224
modf ...................................................244
poly ....................................................347
pow ....................................................353
pow10 ................................................353
rad .....................................................387
rand ....................................................387
randb ..................................................388
randg ..................................................388
sin ......................................................489
sinh ....................................................489
sqrt .....................................................496
srand ..................................................496

Flash, SD
sdspi_debounce ...................................415
sdspi_get_csd ......................................416
sdspi_get_scr .......................................417
sdspi_get_status_reg ............................418
sdspi_getSectorCount ..........................418
sdspi_init_card ....................................419
sdspi_initDevice ..................................420
sdspi_isWriting ...................................421
sdspi_notbusy ......................................421
sdspi_print_dev ...................................422
sdspi_process_command ......................423
sdspi_read_sector ................................424
sdspi_reset_card ..................................425
sdspi_sendingAP .................................426
sdspi_set_block_length .......................427
sdspi_setLED ......................................426
sdspi_write_sector ...............................429
dsypi_WriteContinue ...........................428

Flash, Serial
sf_getPageCount .................................478
sf_getPageSize ....................................478
sf_init .................................................479
sf_initDevice .......................................480
sf_isWriting .......................................481
sf_pageToRAM .....................................481
sf_RAMToPage .....................................482
sf_readDeviceRAM ...............................483
sf_readPage .........................................484
sf_readRAM ..........................................485
sf_writeDeviceRAM .........................486
sf_writePage .......................................487
sf_writePage .......................................488
sfspi_init ..........................................488

Flash, SD
sdspi_debounce ...................................415
sdspi_get_csd ......................................416
sdspi_get_scr .......................................417
sdspi_get_status_reg ............................418
sdspi_getSectorCount ..........................418
sdspi_init_card ....................................419
sdspi_initDevice ..................................420
sdspi_isWriting ...................................421
sdspi_notbusy ......................................421
sdspi_print_dev ...................................422
sdspi_process_command ......................423
sdspi_read_sector ................................424
sdspi_reset_card ..................................425
sdspi_sendingAP .................................426
sdspi_set_block_length .......................427
sdspi_setLED ......................................426
sdspi_write_sector ...............................429
dsypi_WriteContinue ...........................428

Flash, Serial
sf_getPageCount .................................478
sf_getPageSize ....................................478
sf_init .................................................479
sf_initDevice .......................................480
sf_isWriting .......................................481
sf_pageToRAM .....................................481
sf_RAMToPage .....................................482
sf_readDeviceRAM ...............................483
sf_readPage .........................................484
sf_readRAM ..........................................485
sf_writeDeviceRAM .........................486
sf_writePage .......................................487
sf_writePage .......................................488
sfspi_init ..........................................488

Floating-Point Math
acos ........................................................2
acot ........................................................3
acsc ........................................................3
asec ......................................................10
asin .......................................................10
atan ......................................................11
atan2 ....................................................12
ceil .......................................................24
cos .......................................................40
cosh ......................................................40
deg .......................................................43
exp .......................................................76
fabs ......................................................77
floor ......................................................145
fmod ....................................................145
frexp ...................................................152
labs .....................................................212
ldexp ..................................................213
log ......................................................213
log10 ..................................................224
modf ...................................................244
poly ....................................................347
pow ....................................................353
pow10 ................................................353
rad .....................................................387
rand ....................................................387
randb ..................................................388
randg ..................................................388
sin ......................................................489
sinh ....................................................489
sqrt .....................................................496
srand ..................................................496
tan ......................................................518
\textit{tanh} ...........................................519

\textbf{G}

\textbf{Global Positioning System}

gps\_get\_position ..................................182
gps\_get\_utc .......................................183
gps\_ground\_distance ..............................183

\textbf{H}

\textbf{HDLC Protocol (Rabbit 3000, 4000, 5000)}

\textit{HDLCabortE} ..................................186
\textit{HDLCabortF} ..................................186
\textit{HDLCcloseE} ..................................186
\textit{HDLCcloseF} ..................................186
\textit{HDLCdropE} ....................................187
\textit{HDLCdropF} ....................................187
\textit{HDLCerrorE} ..................................187
\textit{HDLCerrorF} ..................................187
\textit{HDLCextClockE} ..............................188
\textit{HDLCextClockF} ..............................188
\textit{HDLCopenE} ....................................189
\textit{HDLCopenF} ....................................189
\textit{HDLCpeekE} ....................................190
\textit{HDLCpeekF} ....................................190
\textit{HDLCreceiveE} .................................191
\textit{HDLCreceiveF} .................................191
\textit{HDLCsendE} ....................................192
\textit{HDLCsendF} ....................................192
\textit{HDLCsendingE} ...............................193
\textit{HDLCsendingF} ...............................193

\textbf{I}

\textbf{I/O}

\textit{BitRdPortE} ...................................18
\textit{BitRdPortI} ...................................19
\textit{BitWrPortE} ...................................20
\textit{BitWrPortI} ...................................21
\textit{RdPortE} ........................................389
\textit{RdPortI} ........................................390
\textit{WrPortE} ........................................539
\textit{WrPortI} ........................................540

\textbf{I2C Protocol}

\textit{i2c\_check\_ack} ...............................206
\textit{i2c\_init} .......................................207
\textit{i2c\_read\_char} ...............................207
\textit{i2c\_send\_ack} ...............................208
\textit{i2c\_send\_nak} ...............................208
\textit{i2c\_start\_tx} .................................209
\textit{i2c\_startw\_tx} ...............................210
\textit{i2c\_stop\_tx} ..................................211
\textit{i2c\_write\_char} .............................211

\textbf{Interrupts}

\textit{GetVectExtern2000} .........................180
\textit{GetVectExtern3000} .........................181
\textit{GetVectIntern} ...............................182
\textit{ipres} .........................................196
\textit{ipset} ..........................................197
\textit{SetVectExtern2000} .........................473
\textit{SetVectExtern3000} .........................474
\textit{SetVectExtern4000} .........................475
\textit{SetVectIntern} ...............................476

\textbf{L}

\textbf{Logging Subsystem}

\textit{log\_clean} ...................................214
\textit{log\_close} ...................................215
\textit{log\_condition} ..............................216
\textit{log\_format} ...................................217
\textit{log\_map} .......................................218
\textit{log\_next} .....................................219
\textit{log\_open} .....................................220
\textit{log\_prev} .....................................221
\textit{log\_put} .......................................222
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_seek</td>
<td>223</td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>md5_append</td>
<td>235</td>
</tr>
<tr>
<td>md5_finish</td>
<td>236</td>
</tr>
<tr>
<td>md5_init</td>
<td>235</td>
</tr>
<tr>
<td>MicroC/OS-II</td>
<td></td>
</tr>
<tr>
<td>OSSemAccept</td>
<td>294</td>
</tr>
<tr>
<td>OSSemCreate</td>
<td>295</td>
</tr>
<tr>
<td>OSSemDel</td>
<td>295</td>
</tr>
<tr>
<td>OSSemPend</td>
<td>295</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>296</td>
</tr>
<tr>
<td>OSSemQuery</td>
<td>297</td>
</tr>
<tr>
<td>OSOSQDel</td>
<td>286</td>
</tr>
<tr>
<td>OS_ENTER_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OS_EXIT_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OSFlagAccept</td>
<td>259</td>
</tr>
<tr>
<td>OSFlagCreate</td>
<td>261</td>
</tr>
<tr>
<td>OSFlagDel</td>
<td>262</td>
</tr>
<tr>
<td>OSFlagPend</td>
<td>263</td>
</tr>
<tr>
<td>OSFlagPost</td>
<td>265</td>
</tr>
<tr>
<td>OSFlagQuery</td>
<td>266</td>
</tr>
<tr>
<td>OSInit</td>
<td>267</td>
</tr>
<tr>
<td>OSMboxAccept</td>
<td>267</td>
</tr>
<tr>
<td>OSMboxCreate</td>
<td>268</td>
</tr>
<tr>
<td>OSMboxDel</td>
<td>269</td>
</tr>
<tr>
<td>OSMboxPend</td>
<td>270</td>
</tr>
<tr>
<td>OSMboxPost</td>
<td>271</td>
</tr>
<tr>
<td>OSMboxPostOpt</td>
<td>272</td>
</tr>
<tr>
<td>OSMboxQuery</td>
<td>273</td>
</tr>
<tr>
<td>OSMemCreate</td>
<td>274</td>
</tr>
<tr>
<td>OSMemGet</td>
<td>275</td>
</tr>
<tr>
<td>OSMemPut</td>
<td>276</td>
</tr>
<tr>
<td>OSMemQuery</td>
<td>277</td>
</tr>
<tr>
<td>OSMutexAccept</td>
<td>278</td>
</tr>
<tr>
<td>OSMutexCreate</td>
<td>279</td>
</tr>
<tr>
<td>OSMutexDel</td>
<td>280</td>
</tr>
<tr>
<td>OSMutexPend</td>
<td>281</td>
</tr>
<tr>
<td>OSMutexPost</td>
<td>282</td>
</tr>
<tr>
<td>OSMutexQuery</td>
<td>283</td>
</tr>
<tr>
<td>OSQAccept</td>
<td>284</td>
</tr>
<tr>
<td>OSQCreate</td>
<td>285</td>
</tr>
<tr>
<td>OSQFlush</td>
<td>287</td>
</tr>
<tr>
<td>OSQPend</td>
<td>288</td>
</tr>
<tr>
<td>OSQPost</td>
<td>289</td>
</tr>
<tr>
<td>OSQPostFront</td>
<td>290</td>
</tr>
<tr>
<td>OSQPostOpt</td>
<td>291</td>
</tr>
<tr>
<td>OSQuery</td>
<td>292</td>
</tr>
<tr>
<td>OSSchedLock</td>
<td>293</td>
</tr>
<tr>
<td>OSSchedUnlock</td>
<td>293</td>
</tr>
<tr>
<td>OSSemAccept</td>
<td>294</td>
</tr>
<tr>
<td>OSSemCreate</td>
<td>295</td>
</tr>
<tr>
<td>OSSemDel</td>
<td>295</td>
</tr>
<tr>
<td>OSSemPend</td>
<td>295</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>296</td>
</tr>
<tr>
<td>OSSemQuery</td>
<td>297</td>
</tr>
<tr>
<td>OSOSQDel</td>
<td>286</td>
</tr>
<tr>
<td>OS_ENTER_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OS_EXIT_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OSFlagAccept</td>
<td>259</td>
</tr>
<tr>
<td>OSFlagCreate</td>
<td>261</td>
</tr>
<tr>
<td>OSFlagDel</td>
<td>262</td>
</tr>
<tr>
<td>OSFlagPend</td>
<td>263</td>
</tr>
<tr>
<td>OSFlagPost</td>
<td>265</td>
</tr>
<tr>
<td>OSFlagQuery</td>
<td>266</td>
</tr>
<tr>
<td>OSInit</td>
<td>267</td>
</tr>
<tr>
<td>OSMboxAccept</td>
<td>267</td>
</tr>
<tr>
<td>OSMboxCreate</td>
<td>268</td>
</tr>
<tr>
<td>OSMboxDel</td>
<td>269</td>
</tr>
<tr>
<td>OSMboxPend</td>
<td>270</td>
</tr>
<tr>
<td>OSMboxPost</td>
<td>271</td>
</tr>
<tr>
<td>OSMboxPostOpt</td>
<td>272</td>
</tr>
<tr>
<td>OSMboxQuery</td>
<td>273</td>
</tr>
<tr>
<td>OSMemCreate</td>
<td>274</td>
</tr>
<tr>
<td>OSMemGet</td>
<td>275</td>
</tr>
<tr>
<td>OSMemPut</td>
<td>276</td>
</tr>
<tr>
<td>OSMemQuery</td>
<td>277</td>
</tr>
<tr>
<td>OSMutexAccept</td>
<td>278</td>
</tr>
<tr>
<td>OSMutexCreate</td>
<td>279</td>
</tr>
<tr>
<td>OSMutexDel</td>
<td>280</td>
</tr>
<tr>
<td>OSMutexPend</td>
<td>281</td>
</tr>
<tr>
<td>OSMutexPost</td>
<td>282</td>
</tr>
<tr>
<td>OSMutexQuery</td>
<td>283</td>
</tr>
<tr>
<td>OSQAccept</td>
<td>284</td>
</tr>
<tr>
<td>OSQCreate</td>
<td>285</td>
</tr>
<tr>
<td>OSQFlush</td>
<td>287</td>
</tr>
<tr>
<td>OSQPend</td>
<td>288</td>
</tr>
<tr>
<td>OSQPost</td>
<td>289</td>
</tr>
<tr>
<td>OSQPostFront</td>
<td>290</td>
</tr>
<tr>
<td>OSQPostOpt</td>
<td>291</td>
</tr>
<tr>
<td>OSQuery</td>
<td>292</td>
</tr>
<tr>
<td>OSSchedLock</td>
<td>293</td>
</tr>
<tr>
<td>OSSchedUnlock</td>
<td>293</td>
</tr>
<tr>
<td>OSSemAccept</td>
<td>294</td>
</tr>
<tr>
<td>OSSemCreate</td>
<td>295</td>
</tr>
<tr>
<td>OSSemDel</td>
<td>295</td>
</tr>
<tr>
<td>OSSemPend</td>
<td>295</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>296</td>
</tr>
<tr>
<td>OSSemQuery</td>
<td>297</td>
</tr>
<tr>
<td>OSOSQDel</td>
<td>286</td>
</tr>
<tr>
<td>OS_ENTER_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OS_EXIT_CRITICAL</td>
<td>258</td>
</tr>
<tr>
<td>OSFlagAccept</td>
<td>259</td>
</tr>
<tr>
<td>OSFlagCreate</td>
<td>261</td>
</tr>
<tr>
<td>OSFlagDel</td>
<td>262</td>
</tr>
<tr>
<td>OSFlagPend</td>
<td>263</td>
</tr>
<tr>
<td>OSFlagPost</td>
<td>265</td>
</tr>
</tbody>
</table>
### Miscellaneous

- `hexstrtobyte` ...........................................193
- `longjmp` ..................................................224
- `qsort` .....................................................386
- `runwatch` ................................................415
- `setjmp` .....................................................471

### Multitasking

- `CoBegin` ..................................................29
- `CoPause` ..................................................38
- `CoResume` ..................................................38
- `DelayMs` ..................................................43
- `DelaySec` ..................................................44
- `DelayTicks` ...............................................45
- `IntervalMs` ...............................................195
- `IntervalSec` ...............................................195
- `IntervalTick` .............................................196
- `isCoDone` ..................................................199
- `isCoRunning` .............................................199
- `loophead` ...............................................225
- `loopinit` ...................................................225

### Number-to-String Conversion

- `ftoa` .......................................................175
- `htoa` .......................................................194
- `itoa` .......................................................205
- `ltoa` .......................................................226
- `ltoan` .....................................................227
- `utoa` .......................................................527

### Partitions

- `mbr_CreatePartition` ....................................229
- `mbr_EnumDevice` .......................................230
- `mbr_FormatDevice` .....................................231
- `mbr_MountPartition` .....................................232
- `mbr_UnmountPartition` ..................................233
- `mbrValidatePartitions` ................................234

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

- `pwm_init` ..................................................368
- `pwm_set` ...................................................369

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

- `qd_error` ...................................................383
- `qd_init` ...................................................384
- `qd_read` ...................................................385
- `qd_zero` ...................................................385

### Rabbit 3000, 4000

- `disableIObus` ............................................46
- `enableIObus` .............................................69
- `servo_alloc_table` .....................................430
- `servo_closedloop` .....................................430
- `servo_disable_0` .....................................431
- `servo_disable_1` .....................................432
- `servo_enable_0` .....................................433
- `servo_enable_1` .....................................434
- `servo_gear` .............................................435
- `servo_graph` ...........................................437
- `servo_init` .............................................438
- `servo_millirpm2vcmd` ..................................438
- `servo_move_to` .......................................439
- `servo_openloop` ......................................440
- `servo_qd_zero_0` .....................................441
- `servo_qd_zero_1` .....................................441
- `servo_read_table` .....................................442
- `servo_set_coeffs` .....................................443
- `servo_set_pos` ........................................444
- `servo_set_vel` ........................................445
- `servo_stats_reset` ..................................445
servo_torque ...........................................446

**Rabbit 3000, 4000, 5000**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cof_pktEreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktEsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_pktFreceive</td>
<td>29</td>
</tr>
<tr>
<td>cof_pktFsend</td>
<td>30</td>
</tr>
<tr>
<td>cof_serEgetc</td>
<td>31</td>
</tr>
<tr>
<td>cof_serEgets</td>
<td>32</td>
</tr>
<tr>
<td>cof_serEputc</td>
<td>33</td>
</tr>
<tr>
<td>cof_serEputs</td>
<td>34</td>
</tr>
<tr>
<td>cof_serEread</td>
<td>35</td>
</tr>
<tr>
<td>cof_serEwrite</td>
<td>36</td>
</tr>
<tr>
<td>cof_serFgetc</td>
<td>31</td>
</tr>
<tr>
<td>cof_serFgets</td>
<td>32</td>
</tr>
<tr>
<td>cof_serFputc</td>
<td>33</td>
</tr>
<tr>
<td>cof_serFputs</td>
<td>34</td>
</tr>
<tr>
<td>cof_serFread</td>
<td>35</td>
</tr>
<tr>
<td>cof_serFwrite</td>
<td>36</td>
</tr>
<tr>
<td>HDLCabortE</td>
<td>186</td>
</tr>
<tr>
<td>HDLCabortF</td>
<td>186</td>
</tr>
<tr>
<td>HDLCcloseE</td>
<td>186</td>
</tr>
<tr>
<td>HDLCcloseF</td>
<td>186</td>
</tr>
<tr>
<td>HDLCdropE</td>
<td>187</td>
</tr>
<tr>
<td>HDLCdropF</td>
<td>187</td>
</tr>
<tr>
<td>HDLCerrorE</td>
<td>187</td>
</tr>
<tr>
<td>HDLCerrorF</td>
<td>187</td>
</tr>
<tr>
<td>HDLCextClockE</td>
<td>188</td>
</tr>
<tr>
<td>HDLCextClockF</td>
<td>188</td>
</tr>
<tr>
<td>HDLCopenE</td>
<td>189</td>
</tr>
<tr>
<td>HDLCopenF</td>
<td>189</td>
</tr>
<tr>
<td>HDLCpeekE</td>
<td>190</td>
</tr>
<tr>
<td>HDLCpeekF</td>
<td>190</td>
</tr>
<tr>
<td>HDLCreceiveE</td>
<td>191</td>
</tr>
<tr>
<td>HDLCreceiveF</td>
<td>191</td>
</tr>
<tr>
<td>HDLCsendE</td>
<td>192</td>
</tr>
<tr>
<td>HDLCsendF</td>
<td>192</td>
</tr>
<tr>
<td>HDLCsendingE</td>
<td>193</td>
</tr>
<tr>
<td>HDLCsendingF</td>
<td>193</td>
</tr>
</tbody>
</table>

**Rabbit 4000, 5000**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAalloc</td>
<td>47</td>
</tr>
<tr>
<td>DMAcompleted</td>
<td>48</td>
</tr>
<tr>
<td>DMAhandle2chan</td>
<td>49</td>
</tr>
<tr>
<td>DMAioe2mem</td>
<td>50</td>
</tr>
<tr>
<td>DMAioi2mem</td>
<td>52</td>
</tr>
<tr>
<td>DMAloadBufDesc</td>
<td>53</td>
</tr>
<tr>
<td>DMAmatchSetup</td>
<td>54</td>
</tr>
<tr>
<td>DMAmem2ioe</td>
<td>55</td>
</tr>
<tr>
<td>DMAmem2ioi</td>
<td>56</td>
</tr>
<tr>
<td>DMAmem2mem</td>
<td>57</td>
</tr>
<tr>
<td>DMAPoll</td>
<td>58</td>
</tr>
<tr>
<td>DMAPrintBufDesc</td>
<td>59</td>
</tr>
<tr>
<td>DMAPrintRegs</td>
<td>60</td>
</tr>
<tr>
<td>DMAsetBufDesc</td>
<td>61</td>
</tr>
<tr>
<td>DMAsetDirect</td>
<td>62</td>
</tr>
<tr>
<td>DMAsetParameters</td>
<td>63</td>
</tr>
<tr>
<td>DMAstartAuto</td>
<td>64</td>
</tr>
<tr>
<td>DMAstartDirect</td>
<td>65</td>
</tr>
<tr>
<td>DMAstop</td>
<td>66</td>
</tr>
<tr>
<td>DMAstopDirect</td>
<td>67</td>
</tr>
<tr>
<td>DMATimerSetup</td>
<td>67</td>
</tr>
<tr>
<td>DMAunalloc</td>
<td>68</td>
</tr>
<tr>
<td>root2vram</td>
<td>412</td>
</tr>
<tr>
<td>serAdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serAdmaOn</td>
<td>450</td>
</tr>
<tr>
<td>serBdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serBdmaOn</td>
<td>450</td>
</tr>
<tr>
<td>serCdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serCdmaOn</td>
<td>450</td>
</tr>
<tr>
<td>serDdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serDdmaOn</td>
<td>450</td>
</tr>
</tbody>
</table>
serEdmaOff .........................................449
serEdmaOn .........................................450
serFdmaOff .........................................449
serFdmaOn .........................................450
serXdmaOff ........................................449
serXdmaOn .........................................450
vram2root ...........................................528

Real-Time Clock
mktime ...............................................242
mktm ..................................................243
read_rtc ..............................................392
read_rtc_32kHz ...................................392
rte_timezone .......................................414
set32kHzDivider ....................................467
tm_rd ..................................................521
tm_wr ..................................................522
updateTimers .......................................524
use32kHzOsc .........................................524
write_rtc .............................................534

S
Serial Communication
cof_serAgetc .........................................31
cof_serAgets .........................................32
cof_serAputc .........................................33
cof_serAputs .........................................34
cof_serAread .........................................35
cof_serAwrite ........................................36
cof_serBgetc .........................................31
cof_serBgets .........................................32
cof_serBputc .........................................33
cof_serBputs .........................................34
cof_serBread .........................................35
cof_serBwrite ........................................36
cof_serCgetc .........................................31
cof_serCgets .........................................32
cof_serCputc .........................................33
cof_serCputs .........................................34
cof_serCread .........................................35
cof_serCwrite .........................................36
cof_serDgetc .........................................31
cof_serDgets .........................................32
cof_serDputc .........................................33
cof_serDputs .........................................34
cof_serDread .........................................35
cof_serDwrite .........................................36
cof_serEgetc .........................................31
cof_serEgets .........................................32
cof_serEputc .........................................33
cof_serEputs .........................................34
cof_serEread .........................................35
cof_serEwrite .........................................36
cof_serFgetc .........................................31
cof_serFgets .........................................32
cof_serFputc .........................................33
cof_serFputs .........................................34
cof_serFread .........................................35
cof_serFwrite .........................................36
serAclose ............................................448
serAdatabits .........................................448
serAdmaOff ..........................................449
serAdmAOpen .........................................450
serAflowcontrolOn ..................................452
serAgetc .............................................453
serAgetError .........................................454
serAopen .............................................455
serAparity ...........................................456
serApeek .............................................457
serAputc .............................................458
serAputs .............................................459
serArdFlush .........................................460
serArdFree ..........................................460
serArdUsed ..........................................461
serAread .............................................462
serAwrFlush .........................................463
serAwrFree ..........................................463
serAwrite ............................................464
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>serAwrUsed</td>
<td>465</td>
</tr>
<tr>
<td>serBclose</td>
<td>448</td>
</tr>
<tr>
<td>serBdatabits</td>
<td>448</td>
</tr>
<tr>
<td>serBdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serBdmaOn</td>
<td>450</td>
</tr>
<tr>
<td>serBflowcontrolOn</td>
<td>452</td>
</tr>
<tr>
<td>serBgetc</td>
<td>453</td>
</tr>
<tr>
<td>serBgetError</td>
<td>454</td>
</tr>
<tr>
<td>serBopen</td>
<td>455</td>
</tr>
<tr>
<td>serBparity</td>
<td>456</td>
</tr>
<tr>
<td>serBpeek</td>
<td>457</td>
</tr>
<tr>
<td>serBputc</td>
<td>458</td>
</tr>
<tr>
<td>serBputs</td>
<td>459</td>
</tr>
<tr>
<td>serBrdFlush</td>
<td>460</td>
</tr>
<tr>
<td>serBrdFree</td>
<td>460</td>
</tr>
<tr>
<td>serBrdUsed</td>
<td>461</td>
</tr>
<tr>
<td>serBread</td>
<td>462</td>
</tr>
<tr>
<td>serBwrFlush</td>
<td>463</td>
</tr>
<tr>
<td>serBwrFree</td>
<td>463</td>
</tr>
<tr>
<td>serBwrite</td>
<td>464</td>
</tr>
<tr>
<td>serBwrUsed</td>
<td>465</td>
</tr>
<tr>
<td>serCclose</td>
<td>448</td>
</tr>
<tr>
<td>serCdatabits</td>
<td>448</td>
</tr>
<tr>
<td>serCdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serCdmaOn</td>
<td>450</td>
</tr>
<tr>
<td>serCflowcontrolOn</td>
<td>452</td>
</tr>
<tr>
<td>serCgetc</td>
<td>453</td>
</tr>
<tr>
<td>serCheckParity</td>
<td>447</td>
</tr>
<tr>
<td>serCopen</td>
<td>455</td>
</tr>
<tr>
<td>serCparity</td>
<td>456</td>
</tr>
<tr>
<td>serCpeek</td>
<td>457</td>
</tr>
<tr>
<td>serCputc</td>
<td>458</td>
</tr>
<tr>
<td>serCputs</td>
<td>459</td>
</tr>
<tr>
<td>serCrdFlush</td>
<td>460</td>
</tr>
<tr>
<td>serCrdFree</td>
<td>460</td>
</tr>
<tr>
<td>serCrdUsed</td>
<td>461</td>
</tr>
<tr>
<td>serCread</td>
<td>462</td>
</tr>
<tr>
<td>serCwrFlush</td>
<td>463</td>
</tr>
<tr>
<td>serCwrFree</td>
<td>463</td>
</tr>
<tr>
<td>serCwrite</td>
<td>464</td>
</tr>
<tr>
<td>serCwrite</td>
<td>464</td>
</tr>
<tr>
<td>serCrdUsed</td>
<td>465</td>
</tr>
<tr>
<td>serCclose</td>
<td>448</td>
</tr>
<tr>
<td>serCdmaOff</td>
<td>449</td>
</tr>
<tr>
<td>serCdmaOn</td>
<td>450</td>
</tr>
<tr>
<td>serCflowcontrolOff</td>
<td>451</td>
</tr>
<tr>
<td>serCflowcontrolOn</td>
<td>452</td>
</tr>
<tr>
<td>serEgetc</td>
<td>453</td>
</tr>
<tr>
<td>serEgetError</td>
<td>454</td>
</tr>
<tr>
<td>serEopen</td>
<td>455</td>
</tr>
<tr>
<td>serEparity</td>
<td>456</td>
</tr>
<tr>
<td>serEpeek</td>
<td>457</td>
</tr>
<tr>
<td>serEputc</td>
<td>458</td>
</tr>
<tr>
<td>serEputs</td>
<td>459</td>
</tr>
<tr>
<td>serErdFlush</td>
<td>460</td>
</tr>
<tr>
<td>serErdFree</td>
<td>460</td>
</tr>
</tbody>
</table>
serErrUsed .........................................461
serEread ..............................................462
serEwrFlush ........................................463
serEwrFree ..........................................464
serEwrUsed .........................................465
serEclose ...........................................448
serFdbits ............................................448
serFdmaOff ........................................449
serFdmaOn .........................................450
serFflowcontrolOff ..............................451
serFflowcontrolOn ................................452
serFgetc ..............................................453
serFgetError .......................................454
serFopen .............................................455
serFparity ............................................456
serFpeek .............................................457
serFputc .............................................458
serFputs ..............................................459
serFrdFlush .........................................460
serFrdFree ..........................................460
serFrdUsed .........................................461
serFread ..............................................462
serFwrFlush .........................................463
serFwrFree ..........................................463
serFwrite ............................................464
serFwrdUsed .......................................465
serFdbits ............................................448
serFdmaOff ........................................449
serFdmaOn .........................................450
serFflowcontrolOff ..............................451
serFflowcontrolOn ................................452
serFgetc ..............................................453
serFgetError .......................................454
serFopen .............................................455
serFparity ............................................456
serFpeek .............................................457
serFputc .............................................458
serFputs ..............................................459
serFrdFlush .........................................460
serFrdFree ..........................................460
serFrdUsed .........................................461
serFread ..............................................462
serFwrFlush .........................................463
serFwrFree ..........................................463
serFwrite ............................................464
serFwrdUsed .......................................465
serXdbits ............................................448
serXdmaOff ........................................449
serXdmaOn .........................................450
serXflowcontrolOff ..............................451
serXflowcontrolOn ................................452
serXget ..............................................453
serXgetErr ...........................................454
serXparity ............................................456
serXpeek .............................................457
serXputc .............................................458
serXputs ..............................................459
serXrdFlush .........................................460
serXrdFree ..........................................460
serXread ..............................................462
serXwrFlush .........................................463
serXwrFree ..........................................463
serXwrite ............................................464
serXwrdUsed .......................................465

Serial Packet Driver
cof_pktAreceive ....................................29
cof_pktAsend .......................................30
cof_pktBreceive ....................................29
cof_pktBsend .......................................30
cof_pktCreceive ....................................29
cof_pktCsend .......................................30
cof_pktDreceive ....................................29
cof_pktDsend .......................................30
cof_pktEreceive ....................................29
cof_pktEsend .......................................30
cof_pktFreceive ....................................29
cof_pktFsend .......................................30
pktAclose ..........................................332
pktAgetErrors .....................................332
pktAinitBuffers ...................................333
pktAopen ..........................................334
pktAreceive ........................................336
pktAsend ..........................................337
pktAsending .......................................338
pktAsetParity ......................................338
pktBclose ..........................................332
pktBgetErrors .....................................332
pktBinitBuffers ...................................333
pktBopen ..........................................334
pktBreceive ........................................336
pktBsend ..........................................337
pktBsending .......................................338
pktBsetParity ......................................338
pktCclose ..........................................332
pktCgetErrors .....................................332
pktCinitBuffers ...................................333
pktCopen ............................................334
pktCreceive .........................................336
pktCsend .............................................337
pktCsending ........................................338
pktCsetParity .......................................338
pktDclose ............................................332
pktDgetErrors ......................................332
pktDinitBuffers ...................................333
pktDopen ............................................334
pktDreceive .........................................336
pktDsend ............................................337
pktDsending ........................................338
pktDsetParity .......................................338
pktEclose ............................................332
pktEgetErrors ......................................332
pktEinitBuffers ....................................333
pktEopen ............................................334
pktEreceive .........................................336
pktEsend .............................................337
pktEsending ........................................338
pktEsetParity .......................................338
Servo Control (Rabbit 3000, 4000)
servo_alloc_table .................................430
servo_closedloop .................................430
servo_disable_0 ...................................431
servo_disable_1 ...................................432
servo_enable_0 ...................................433
servo_enable_1 ...................................434
servo_gear .........................................435
servo_graph ........................................437
servo_init ..........................................438
servo_move_to ....................................439
servo_openloop ....................................440
servo_qd_zero_0 ...................................441
servo_qd_zero_1 ...................................441
servo_read_table ..................................442
servo_set_coeffs .................................443
servo_set_pos .....................................444
servo_set_vel .....................................445
servo_stats_reset .................................445
servo_torque ......................................446
SPI
SPIinit ..............................................491
SPIRead .............................................492
SPIWrite ............................................493
SPIWrRd ............................................494
Stdio
getchar .............................................175
gets ...............................................177
kbhit ..............................................212
outchrs .............................................317
outstr ..............................................318
printf ..............................................361
putchar .............................................367
puts ...............................................367
snprintf ..........................................490
sprintf .............................................495
String Manipulation
memchr .............................................237
memcmp .............................................238
mempcpy .............................................239
memmove ............................................240
memset .............................................241
streat ..............................................497
strchr ..............................................498
strcmp .................................................499
strcmpi ...............................................500
strcpy ..................................................501
strcspn ................................................502
strlen ..................................................503
strncat .................................................504
strncmp ...............................................505
strncmpi ..............................................506
strncpy ................................................507
strpbrk ................................................508
strrchr .................................................509
strrsn ..................................................510
strtok ..................................................511
tolower ...............................................523
toupper ...............................................523
GetVectExtern3000 ...............................181
GetVectIntern ....................................182
ipres ....................................................196
ipset ....................................................197
premain ...............................................358
set_cpu_power_mode .............................469
set32kHzDivider ..................................467
setClockModulation .............................468
SetSerialTATxRValues ..........................472
sysResetChain ....................................517
TAT1R_SetValue ....................................520
updateTimers .......................................524
use32kHzOsc ......................................524
useClockDivider .................................525
useClockDivider3000 ............................526
useMainOsc .........................................527

String-to-Number Conversion
atof ......................................................13
atoi ......................................................14
atol ......................................................15
strtod ...................................................512
strtol ...................................................515

System
_GetSysMacroIndex ..............................178
_GetSysMacroValue ..............................179
_sysIsSoftReset ..................................517
cchkHardReset ..................................25
cchkSoftReset ..................................25
cchkWDTO ..........................................26
clockDoublerOff .................................27
clockDoublerOn .................................27
defineErrorHandler ............................42
ext ...................................................76
forceSoftReset ..................................150
get_cpu_frequency ..............................176
getdivider19200 ............................177
GetVectExter2000 ..............................180

User Block
readUserBlock .................................393
readUserBlockArray ..........................394
writeUserBlock ..................................535
writeUserBlockArray ..........................537

VBAT RAM (Rabbit 4000, 5000)
root2vram ......................................412
vram2root .....................................528

Watchdogs
Disable_HW_WDT ...............................45
Enable_HW_WDT ..................................69
hitwd ...............................................194
VdGetFreeWd ....................................529
VdHitWd ............................................530
VdInit ..............................................530
VdReleaseWd .....................................531
1. Function Descriptions

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

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

```c
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**

```c
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
void AESdecrypt4x4( char far * expandedkey, char far * crypt, char far * plain );

DESCRIPTION

Decrypts a block of data using an implementation of the Rijndael AES cipher with a 128-bit key and block size.

The encrypted block of data may be overwritten by the decrypted block of data.

PARAMETERS

- **expandedkey** A set of round keys (generated by AESexpandKey4()) from a 16-byte (128 bit) key.
  
  Total of 176 bytes (44 longwords)

  Note: when using an AESstreamState structure (e.g. “state”) then call this function using:

  AESdecrypt4x4(state->expanded_key, plain, crypt);

- **crypt** A block of 16 bytes of ciphertext to be decrypted; “crypt” and “plain” may point to the same place.

- **plain** A block of 16 bytes of resulting plaintext data; “crypt” and “plain” may point to the same place.

LIBRARY

AES_CORE.LIB
int AESdecryptStream4x4_CBC(AESstreamState * state, long message,
long output, unsigned int count);

DESCRIPTION
Perform an AES-CBC decryption operation.
See Samples\Crypt\AES_STREAMTEST.C for a sample program and a detailed explanation of the encryption/decryption process.

PARAMETERS
- **state**: The AESstreamState structure, initialized via AESinitStream4x4().
  This memory must be allocated in the program code before calling AESdecryptStream4x4_CBC():
  ```c
  static AESstreamState decrypt_state;
  ```
- **message**: Cipher-text message (an xmem buffer)
- **output**: Output buffer, for return of decrypted text (in xmem). Must be as large as the cipher-text buffer. May be the same as the cipher-text buffer.
- **count**: Length of the message. Must a multiple of _AES_CBC_BLK_SZ_ (16).

RETURN VALUE
0 on success, non-zero on failure

LIBRARY
AES_CORE.LIB
AESencrypt4x4

void AESencrypt4x4( char far * expandedkey, char far * plain,
char far * crypt );

DESCRIPTION
Encrypts a block of data using an implementation of the Rijndael AES cipher with 128-bit key and block size. The block of data may be overwritten by the encrypted block of data.

PARAMETERS
expandedkey A set of round keys (generated by AESexpandKey4()) from a 16-byte (128 bit) key.
Total of 176 bytes (44 longwords)
Note: when using an AESstreamState structure (e.g., “state”) then call this function using:
 AESencrypt4x4(state->expanded_key, plain, crypt);
plain A block of 16 bytes of data to be encrypted; “crypt” and “plain” may point to the same place.
crypt A block of 16 bytes of resulting encrypted data; “crypt” and “plain” may point to the same place.

RETURN VALUE
None.

LIBRARY
AES_CORE.LIB
int AESencryptStream4x4_CBC( AESstreamState * state, long message, 
    long output, unsigned int count);

DESCRIPTION
Perform an AES-CBC encryption operation on XMEM data. Encryption is not “in-place.”
See Samples\Crypt\AES_STREAMTEST.C for a sample program and a detailed explanation of the encryption/decryption process.

PARAMETERS

state An AES stream state structure, initialized via AESinitStream4x4().
This memory must be allocated in the program code before calling
AESencryptStream():
static AESstreamState encrypt_state;

message The message in plaintext (an xmem buffer)

output The output buffer, for return of encrypted text (in xmem), must be as large
as the plaintext buffer, and may be the same as the plaintext buffer.

count The length of the message. Must be a multiple of _AES_CBC_BLK_SZ_ (16).

RETURN VALUE
0 on success, non-zero on failure (count was not multiple of 16)

LIBRARY
AES_CORE.LIB
void AESexpandKey4( char far * expanded, char far * key );

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

This function is specific to AES with 128-bit key. See AESexpandKey() for a more general function (available with Rabbit Embedded Security Pack).

PARAMETERS
expanded A buffer for storing the expanded key. The size of the expanded key, for a 128-bit key, is 176 bytes. Other key sizes are not supported by this function.

Note: when using an AESstreamState structure (e.g., “state”) then call this function using:

AESexpandKey4(state->expanded_key, key);

key The cipher key, 16 bytes

RETURN VALUE
None.

LIBRARY
AES_CORE.LIB
void AESinitStream4x4( AESstreamState far * state, char far * key, char far * init_vector);

DESCRIPTION
Sets up a stream state structure to begin encrypting or decrypting a stream using AES with a 128-bit key and block size. A particular stream state 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 AESinitStream4x4().

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 and key. For security, it is very important never to use the same initialization vector twice with the same key.

RETURN VALUE
None.

LIBRARY
AES_CORE.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**

```c
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`
**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:

<table>
<thead>
<tr>
<th>Returned Value (in Radians)</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$angle$</td>
<td>$x \neq 0, y \neq 0$</td>
</tr>
<tr>
<td>$\pi/2$</td>
<td>$x = 0, y &gt; 0$</td>
</tr>
<tr>
<td>$-\pi/2$</td>
<td>$x = 0, y &lt; 0$</td>
</tr>
<tr>
<td>0</td>
<td>$x &gt; 0, y = 0$</td>
</tr>
<tr>
<td>$\pi$</td>
<td>$x &lt; 0, y = 0$</td>
</tr>
</tbody>
</table>

**Library**

`MATH.LIB`

**See Also**

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

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

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

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

**DESCRIPTION**

ANSI string to float conversion (UNIX compatible).

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

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

**PARAMETERS**

- `sptr` String to convert.

**RETURN VALUE**

The converted floating value.

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

**LIBRARY**

`STRING.LIB`

**SEE ALSO**

`atoi, atol, strtod`
**atoi**

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

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

**DESCRIPTION**

ANSI string to integer conversion (UNIX compatible).

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

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

**PARAMETERS**

- sptr String to convert.

**RETURN VALUE**

The converted integer value.

**LIBRARY**

STRING.LIB

**SEE ALSO**

atol, atof, strtod
**atol**

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

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

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

**DESCRIPTION**

ANSI string to long conversion (UNIX compatible).

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

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

**PARAMETERS**

`sptr`  
String to convert.

**RETURN VALUE**

The converted long integer value.

**LIBRARY**

`STRING.LIB`

**SEE ALSO**

`atoi, atof, strtod`
Dynamic C Functions

**bit**

```c
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:

\[ \left(\left(\ast\left(\text{long } \ast\right)\text{address} \gg \text{bit}\right) \& 1 \right) \]

**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`
Dynamic C Functions

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:

\[ \left(\ast\left(\text{long } \ast\text{address}\right)\gg\text{bit}\right) \& 1 \]

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

```c
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
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
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)
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**

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

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

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

**LIBRARY**
SYS.LIB

**SEE ALSO**
- `chkSoftReset`, `chkWDTO`, `_sysIsSoftReset`

---

**chkSoftReset**

```c
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`
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

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

**DESCRIPTION**

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

**LIBRARY**

SYS.LIB

**SEE ALSO**

`clockDoublerOff`

### clockDoublerOff

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

**DESCRIPTION**

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

**LIBRARY**

SYS.LIB

**SEE ALSO**

`clockDoublerOn`
CloseInputCompressedFile

```c
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

```c
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

```c
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

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

**DESCRIPTION**

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

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

**PARAMETERS**

- `buffer` A buffer for the packet to be written into.
- `buffer_size` Length of the buffer.

**RETURN VALUE**

- >0: The number of bytes in the received packet on success.
- 0: No new packets have been received.
- -1: The packet is too large for the given buffer.
- -2: A needed test_packet function is not defined.

**LIBRARY**

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

DESCRIPTION

Initiates the sending of a packet of data. The function will exit when the packet is finished trans-
mitting.

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

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**

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

**DESCRIPTION**

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

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

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

**RETURN VALUE**

An integer with the character read into the low byte.

**LIBRARY**

`RS232.LIB`

**EXAMPLE**

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

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

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

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

PARAMETERS
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**

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

**DESCRIPTION**

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

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

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

**PARAMETERS**

- `c` Character to write.

**LIBRARY**

`RS232.LIB`

**EXAMPLE**

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

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

**DESCRIPTION**

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

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

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

**PARAMETERS**

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

**LIBRARY**

RS232.LIB

**EXAMPLE**

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

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

**DESCRIPTION**

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

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

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

**PARAMETERS**

- **data**   Data structure into which characters are read.
- **length** The number of characters to read into `data`.
- **tmout**  Millisecond wait period to allow between characters before timing out.

**RETURN VALUE**

Number of characters read into `data`.

**LIBRARY**

RS232.LIB

**EXAMPLE**

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

DESCRIPTION

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

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

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

PARAMETERS

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 Rabbit";
    serAopen(19200);
    loopinit();
    while (1) {
        loophead();
        costate {
            wfd cof_serAwrite(s, strlen(s));
        }
    }
    serAclose();
}
CompressFile

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

DESCRIPTION

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

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

PARAMETERS

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 encoun-
    tered.

PARAMETERS

    p        Address of costatement

LIBRARY

    COSTATE.LIB
CoResume

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

**DESCRIPTION**

Resume execution of a costatement that has been paused.

**PARAMETERS**

- `p` Address of costatement

**LIBRARY**

```
COSTATE.LIB
```
**cos**

```c
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**

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

**DESCRIPTION**
Computes the hyperbolic cosine of real float value \( x \). This function 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`
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

```c
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:

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

**PARAMETERS**

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

**LIBRARY**

`SYS.LIB`
### deg

```c
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` and `DelayMs`

---

### DelayMs

```c
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.

Note that milliseconds timing starts immediately, without waiting for the current millisecond to elapse. In the case that the current millisecond is just about to end, the perceived elapsed time may be as much as 1 millisecond shorter than the requested delay.

**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

```c
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.

Note that seconds timing starts immediately, without waiting for the current second to elapse. In the case that the current second is just about to end, the perceived elapsed time may be as much as 1 second shorter than the requested delay. For more precise delays of up to 24 days duration, consider using `DelayMs()` instead of `DelaySec()`.

**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**

```c
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.

Note that tick timing starts immediately, without waiting for the current tick to elapse. In the case that the current tick is just about to end, the perceived elapsed time may be as much as 1 tick shorter than the requested delay.

**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**

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

**DESCRIPTION**

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

**LIBRARY**

`SYS.LIB`
void disableIObus( void );

DESCRIPTION

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

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

This function is non-reentrant.

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

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

LIBRARY

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

SEE ALSO

enableIObus
DMAalloc

dma_chan_t DMAalloc( char channel_mask, int highest );

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

PARAMETERS
channel_mask  Mask of all the acceptable channels to choose from.
highest      Bool indicating whether to search for an available channel from 8 or from 0.

RETURN VALUE
Returns a handle to a DMA channel if one is available. If none are available it returns DMA_CHANNEL_NONE.

LIBRARY
DMA.LIB

SEE ALSO
DMAunalloc, DMAhandle2chan
int DMAcompleted( dma_chan_t handle, unsigned int * len );

DESCRIPTION

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

PARAMETERS

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

RETURN VALUE

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

LIBRARY

DMA.LIB

SEE ALSO

DMAstop
int DMAhandle2chan( dma_chan_t handle );

DESCRIPTION

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

PARAMETER

handle Handle to convert to channel number

RETURN VALUE

0-7: Valid channel number
DMA_CHANNEL_NONE: The channel is invalid

LIBRARY

DMA.LIB

SEE ALSO

DMA alloc, DMA unalloc
int DMAioe2mem( dma_chan_t handle, dma_addr_t dest, unsigned int src, unsigned int len, unsigned int flags );

DESCRIPTION

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

PARAMETERS

handle  Handle for channel to use in transfer

dest    Memory destination address

src     External I/O location source address

len     Length to send (cannot equal zero)

flags   Various flag options.

• DMA_F_REPEAT indicates that the transfer will be a cycle
• DMA_F_INTERRUPT indicates an interrupt will be triggered at the completion of the transfer. The interrupt vector and function must be set up in the user’s code.
• DMA_F_LAST_SPECIAL (only for Ethernet or HDLC peripherals)
  Internal Source: Status byte written to initial buffer descriptor before last data.
  Internal Destination: Last byte written to offset address for frame termination.
  All Others: no effect.
• DMA_F_SRC_DEC only for transfers with memory source. Indicates the source address should be decremented.
• DMA_F_DEST_DEC only for transfers with memory destination. Indicates the destination address should be incremented.
• DMA_F_STOP_MATCH indicates whether or not to stop the dma transfer when a character is reached. The match byte and mask should have previously been set by calling the DMAmatchSetup() function.
• DMA_F_TIMER indicates the DMA timer will be used. The divisor should have already been set by calling the DMAtimerSetup() function.
• DMA_F_TIMER_1BPR indicates that the timed transfers will send one byte per request instead of the entire descriptor
DMAioe2mem (cont’d)

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

- DMA_F_PD2, DMA_F_PE2, DMA_F_PE6, DMA_F_PD3,
  DMA_F_PE3, DMA_F_PE7

The following flags indicate the polarity of the gating signal:

- DMA_F_FALLING (default), DMA_F_RISING, DMA_F_LOW,
  DMA_F_HIGH

RETURN VALUE

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

LIBRARY

DMA.LIB

SEE ALSO

DMAmem2mem, DMAcompleted, DMAstop
int DMAioi2mem( dma_chan_t handle, dma_addr_t dest, unsigned int src,
               unsigned int len, unsigned int flags );

DESCRIPTION

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

PARAMETERS

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

RETURN VALUE

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

LIBRARY

   DMA.LIB

SEE ALSO

   DMAmem2mem, DMAcompleted, DMAstop
void DMAloadBufDesc( int dmaChannel, dma_addr_t * bufPtr );

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

PARAMETERS
- **dmaChannel**: DMA channel number to load
- **bufPtr**: Pointer to variable containing physical address of DMA buffer

LIBRARY
DMA.LIB

SEE ALSO
- DMAsetBufDesc, DMAsetDirect
int DMAmatchSetup( dma_chan_t handle, int mask, int byte );

**DESCRIPTION**

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

**PARAMETERS**

- **handle**
  Handle for the DMA channel.

- **mask**
  Mask for termination byte (parameter 3). A value of all zeros disables the termination byte match feature. A value of all ones uses the full termination byte for comparison.

- **byte**
  Byte that, if matched, will terminate the buffer.

**LIBRARY**

DMA.LIB

**SEE ALSO**

DMAmem2mem, DMAtimerSetup
int DMAmem2ioe( dma_chan_t handle, unsigned int dest, dma_addr_t src,
               unsigned int len, unsigned int flags );

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

PARAMETERS
- **handle**: Handle for channel to use in transfer
- **dest**: External I/O destination address
- **src**: Memory location source
- **len**: Length to send (cannot equal zero)
- **flags**: Various flag options. See DMAioe2mem() for a full list of flags and their descriptions.

RETURN VALUE
- **0**: Success
- **EINVAL**: Invalid handle
- **EBUSY**: Resources are busy

LIBRARY
DMA.LIB

SEE ALSO
- DMAmem2mem, DMAcompleted, DMAstop
int DMAmem2ioi( dma-chan_t handle, unsigned int dest, dma-addr_t src,
         unsigned int len, unsigned int flags );

DESCRIPTION

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

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Handle for channel to use in transfer</td>
</tr>
<tr>
<td>dest</td>
<td>Internal I/O destination address</td>
</tr>
<tr>
<td>src</td>
<td>Memory location source</td>
</tr>
<tr>
<td>len</td>
<td>Length to send (cannot equal zero)</td>
</tr>
<tr>
<td>flags</td>
<td>Various flag options. See DMAioe2mem() for a full list of flags and their descriptions.</td>
</tr>
</tbody>
</table>

RETURN VALUE

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

LIBRARY

DMA.LIB

SEE ALSO

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

DESCRIPTION

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

PARAMETERS

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

RETURN VALUE

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

LIBRARY

DMA.LIB

SEE ALSO

DMAcompleted, DMAstop
word DMApoll( int dmaChannel, word * bufCount );

DESCRIPTION

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

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

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

PARAMETERS

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

RETURN VALUE

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

LIBRARY

DMA.LIB

SEE ALSO

DMAloadBufDesc, DMAsetDirect
DMAprintBufDesc

void DMAprintBufDesc( void * dr, long dp );

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

PARAMETERS
  dr  Pointer to buffer descriptor in root memory.
  dp  Address of buffer descriptor in physical memory.

LIBRARY
DMA.LIB

SEE ALSO
DMAprintRegs
**DMAprintRegs**

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

**DESCRIPTION**

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

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

**PARAMETERS**

- **chan**  
  Channel number to print

- **masters**  
  A bool to determine whether or not to print out the master registers shared between all channels

**LIBRARY**

DMA.LIB

**SEE ALSO**

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

DESCRIPTION

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

PARAMETERS

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

RETURN VALUE

    0: Success
    -EINVAL: Error

LIBRARY

    DMA.LIB

SEE ALSO

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

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

PARAMETERS

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

LIBRARY
DMA.LIB

SEE ALSO
DMAloadBufDesc, DMAsetBufDesc
int DMAsetParameters( unsigned int transfer_pri, unsigned int interrupt_pri, unsigned int inter_dma_pri, unsigned int chunkiness, unsigned int min_cpu_pct );

DESCRIPTION

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

<table>
<thead>
<tr>
<th>chunkiness</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_cycles</td>
<td>11</td>
<td>15</td>
<td>19</td>
<td>23</td>
<td>39</td>
<td>71</td>
<td>135</td>
<td>263</td>
</tr>
</tbody>
</table>

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

\[
\text{cpu free time} = \frac{(\text{CPU}_{cycles} \cdot \text{min}_{cpu\_pct})}{(100 - \text{min}_{cpu\_pct})}
\]

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

PARAMETERS

**transfer_pri**     DMA transfer priority (0, 1, 2 or 3), transfers can occur when the CPU interrupt priority is less than or equal to this value.

**interrupt_pri**   DMA interrupt priority (0, 1, 2, or 3); a value of 0 will disable the DMA interrupts.

**inter_dma_pri**   Relative prioritization amongst the DMA channels. It is one of the following constants:

- DMA_IDP_FIXED - fixed priorities, with higher channel numbers taking precedence;
- DMA_IDP_ROTATE_FINE - priorities are rotated after every byte transferred;
- DMA_IDP_ROTATE_COARSE - priorities rotated after every transfer request, the size of which is determined by the “chunkiness” parameter.

**chunkiness**      Maximum transfer burst size. Allowed values are 1, 2, 3, 4, 8, 16, 32, or 64. Other numbers will be rounded down to the nearest allowed value.
DMAsetParameters (cont’d)

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

RETURN VALUE

0: Success
-EINVAL: for an error

LIBRARY

DMA.LIB

DMAstartAuto

void DMAstartAuto( int channel );

DESCRIPTION

This function is defined to the following:

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

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

PARAMETER

channel  DMA channel (obtainable through DMAhandle2chan())

LIBRARY

DMA.LIB

SEE ALSO

DMAstartDirect, DMAstopDirect
void DMAstartDirect( int channel );

DESCRIPTION
This function is defined to the following:

    WrPortI(DMCSR, NULL, 1 << channel);

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

PARAMETER
channel        DMA channel (obtainable through DMAhandle2chan())

LIBRARY
DMA.LIB

SEE ALSO
DMAstartAuto, DMAstopDirect
int DMAstop( dma_chan_t handle );

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

PARAMETER
Handle for channel to stop.

RETURN VALUE
0: Success
~EINVAL: Invalid handle

LIBRARY
DMA.LIB

SEE ALSO
DMAcompleted, DMAstopDirect
**DMAstopDirect**

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

**DESCRIPTION**
This function is defined to the following:

```c
WrPortI(DMHR, NULL, 1 << channel);
```

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

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

**LIBRARY**
DMA.LIB

**SEE ALSO**
- DMAstartAuto, DMAstartDirect

---

**DMAtimerSetup**

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

**DESCRIPTION**
This function sets up the DMA 16-bit divisor. To use the divisor, the DMA_F_TIMER flag must be passed to the transfer function.

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

**LIBRARY**
DMA.LIB

**SEE ALSO**
- DMAmem2mem, DMAmatchSetup
int DMAunalloc( dma_chan_t handle );

DESCRIPTION

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

PARAMETER

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

RETURN VALUE

0: Success
-EINVAL: Error

LIBRARY

DMA.LIB

SEE ALSO

DMAalloc, DMAhandle2chan
Dynamic C Functions

Enable_HW_WDT

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

**DESCRIPTION**

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

**LIBRARY**

SYS.LIB

---

enableIObus

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

**DESCRIPTION**

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

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

This function is non-reentrant.

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

If the macro `PORTA_AUX_IO` has been previously defined, this function should not be called.

**LIBRARY**

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

**SEE ALSO**

disableIObus
errlogGetHeaderInfo

root char* errlogGetHeaderInfo( void );

DESCRIPTION

Reads the error log header and formats the output.

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

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

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

RETURN VALUE

A null terminated string containing the header information:

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

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

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

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

LIBRARY

ERRORS.LIB
### errlogGetNthEntry

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

**DESCRIPTION**

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

**PARAMETERS**

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

**RETURN VALUE**

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

**LIBRARY**

- `ERRORS.LIB`

### errlogFormatEntry

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

**DESCRIPTION**

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

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

**RETURN VALUE**

The null terminated string described above.

**LIBRARY**

- `ERRORS.LIB`
errlogFormatRegDump

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

DESCRIPTION

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

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

RETURN VALUE

The null terminated string described above.

LIBRARY

`ERRORS.LIB`

---

errlogFormatStackDump

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

DESCRIPTION

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

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

RETURN VALUE

The null terminated string describe above.

LIBRARY

`ERRORS.LIB`
errlogGetMessage

root char * errlogGetMessage( void );

DESCRIPTION

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

RETURN VALUE

A null terminated string.

LIBRARY
ERRORS.LIB

errlogReadHeader

root int errlogReadHeader( void );

DESCRIPTION

Reads error log header into the structure errlogInfo.

RETURN VALUE

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

LIBRARY
ERRORS.LIB
**error_message**

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

**DESCRIPTION**

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

**PARAMETER**

- `message_index`  
  Positive or negative value of error return code.

**RETURN VALUE**

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

**LIBRARY**

`ERRORS.LIB`
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:

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

RETURN VALUE

Runtime error code passed to it.

LIBRARY

ERRORS.LIB

SEE ALSO

defineErrorHandler
### exit

```c
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

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

DESCRIPTION

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

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

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

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

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

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

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

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

PARAMETERS

flags

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

- FDDF_MOUNT_PART_0: Mount specified partition
- FDDF_MOUNT_PART_1:
- FDDF_MOUNT_PART_2:
- FDDF_MOUNT_PART_3:
- FDDF_MOUNT_PART_ALL: Mount all partitions
- FDDF_MOUNT_DEV_0: Apply to specified device
- FDDF_MOUNT_DEV_1:
- FDDF_MOUNT_DEV_2:
- FDDF_MOUNT_DEV_3:
- FDDF_MOUNT_DEV_ALL: Apply to all available devices
- FDDF_NO_RECOVERY: Use no recovery if fails first time
- FDDF_COND_DEV_FORMAT: Format device if unformatted
- FDDF_COND_PART_FORMAT: Format partition if unformatted
- FDDF_UNCOND_DEV_FORMAT: Format device unconditionally
- FDDF_UNCOND_PART_FORMAT: Format partition unconditionally

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

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

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

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

LIBRARY

FAT.LIB

SEE ALSO

fat_EnumDevice, fat_EnumPartition, fat_MountPartition
fat_Close

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

**DESCRIPTION**

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

**PARAMETERS**

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

**RETURN VALUE**

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

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

**LIBRARY**

FAT.LIB

**SEE ALSO**

fat_Open, fat_OpenDir
fat_CreateDir

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

DESCRIPTION

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

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

PARAMETERS

part Handle for the partition being used.

dirname Pointer to the full path name of the directory to be created.

RETURN VALUE

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

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

LIBRARY

FAT.LIB

SEE ALSO

fat_ReadDir, fat_Status, fat_Open, fat_CreateFile
int fat_CreateFile( fat_part * part, char * filename, long alloc_size, FATfile * file );

DESCRIPTION

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

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

Valid filenames are limited to an 8 character filename and 3 character extension separated by a period; this is commonly known as the “8.3” format. Examples include but are not limited to “12345678.123”, “filename.txt”, and “webpage1.htm”.

PARAMETERS

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

RETURN VALUE

0: success.
-EINVAL: part, filename, alloc_size, or file contain invalid values.
-ENOENT: the parent directory does not exist.
-ENOSPC: no allocatable sectors were found.
-EPERM: write-protected, trying to create a file on a read-only partition.
-EBUSY: the device is busy (non-blocking mode only).
-EFSTATE: if non-blocking, but a previous sequence of calls to this function (of fat_CreateFile) has not completed but you are trying to create a different file or directory. You must complete the sequence of calls for each file or directory i.e. keep calling until something other than -EBUSY is returned. This code is only returned if you pass a NULL file pointer, or if the file pointer is not NULL and the referenced file is already open.
-EPATHSTR: Bad file/directory path string. Valid filenames are limited to the 8.3 format.
Other negative values indicate I/O error, etc.
fat_CreateFile (cont’d)

LIBRARY

FAT.LIB

SEE ALSO

fat_Open, fat_ReadDir, fat_Write

fat_CreateTime

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

DESCRIPTION

This function puts the creation date and time of the entry into the system time structure \( t \). The function does not fill in the \( \text{tm}_\text{wday} \) field in the system time structure.

PARAMETERS

- entry: Pointer to a directory entry
- t: Pointer to a system time structure

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_ReadDir, fat_Status, fat_LastAccess, fat_LastWrite
int fat_Delete( fat_part *part, int type, char *name );

DESCRIPTION

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

PARAMETERS

part Handle for the partition being used.

type Must be a FAT file (FAT_FILE) or a FAT directory (FAT_DIR), depending on what is to be deleted.

name Pointer to the full path name of the file/directory to be deleted.

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_Open, fat_OpenDir, fat_Split, fat_Truncate, fat_Close
fat(EnumDevice)

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

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

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

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

PARAMETERS

driver Pointer to an initialized driver structure set up during the initialization of
the storage device driver.

dev Pointer to the device structure to be filled in.

devnum Physical device number of the device.

sig Pointer to a unique signature string. Note that this value must remain the
same between resets.

norecovery Boolean flag - set to True to ignore power-recovery data. True is any value
except zero.

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

LIBRARY
FAT.LIB

SEE ALSO
fat_AutoMount, fat_Init, fat_EnumPartition
fat_NumPartition

int fat_NumPartition( mbr_dev *dev, int pnum, fat_part *part );

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

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

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

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

LIBRARY
    FAT.LIB

SEE ALSO
    fat_NumDevice, fat_FormatPartition, fat_MountPartition
fat_FileSize

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

DESCRIPTION

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

PARAMETERS

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

RETURN VALUE

  0: success.
  -EINVAL: file is invalid.

LIBRARY

  FAT.LIB

SEE ALSO

  fat_Open, fat_Seek
**fat_FormatDevice**

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

**DESCRIPTION**

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

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

**PARAMETERS**

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

**RETURN**

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

**LIBRARY**

FAT.LIB

**SEE ALSO**

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

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

**DESCRIPTION**

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

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

**PARAMETERS**

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

**RETURN VALUE**

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

**LIBRARY**

FAT.LIB

**SEE ALSO**

`fat_AutoMount`, `fat_Init`, `fat_FormatDevice`, `fat_EnumDevice`, `fat_PartitionDevice`, `fat_EnumPartition`
int fat_Free( fat_part *part );

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

PARAMETERS
part Handle to the partition.

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

LIBRARY
FAT.LIB

SEE ALSO
fat_EnumPartition, fat_MountPartition
int fat_GetAttr( FATfile *file );

DESCRIPTION

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

• FATATTR_READ_ONLY - The file can not be modified.
• FATATTR_HIDDEN - The file is not visible when doing normal operations.
• FATATTR_SYSTEM - This is a system file and should be left alone.
• FATATTR_VOLUME_ID - This is the name of a logical disk.
• FATATTR_DIRECTORY - This is a directory and not a file.
• FATATTR_ARCHIVE - This tells you when the file was last modified.
• FATATTR_LONG_NAME - This is a FAT32 or long file name. It is not supported.

PARAMETERS

file Handle to the open file.

RETURN VALUE

Attributes on success
-EINVAL: invalid file handle.

LIBRARY

FAT.LIB

SEE ALSO

fat_Open, fat_Status
## fat_GetName

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

**DESCRIPTION**

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

**PARAMETERS**

- **entry**  
  Pointer to a directory entry obtained by `fat_Status()`.

- **buf**  
  Pointer to a `char` array that will be filled in. This array must be at least 13 characters long.

- **flags**  
  May be one of the following:
  - 0 - standard format, e.g., `AUTOEXEC.BAT` or `XYZ.GIF`
  - `FAT_LOWERCASE` - standard format, but make lower case.

**RETURN VALUE**

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

**LIBRARY**

`FAT.LIB`

**SEE ALSO**

`fat_ReadDir, fat_Status`
fat_GetPartition

int fat_GetPartition ( fat_part **part, char **file, char *fullpath);

DESCRIPTION

Split a full pathname (e.g., “a:/filename.txt”) into a partition and filename.

Examples (with FAT_USE_FORWARDSLASH defined):

a:/filename.txt > partition A, /filename.txt
/b/filename.txt > partition B, /filename.txt
C:filename.txt > partition C, /filename.txt

Examples (without FAT_USE_FORWARDSLASH defined):

a:\filename.txt > partition A, \filename.txt
\b\filename.txt > partition B, \filename.txt
C:filename.txt > partition C, \filename.txt

PARAMETERS

part Memory location to store a pointer to the fat partition (drive letter).

file Memory location to store a pointer into fullpath (parameter 3) where the
filename begins.

fullpath Pathname to parse.

RETURN VALUE

0: Success
-EINVAL: unable to parse <fullpath>

LIBRARY

FAT.LIB
fat_Init

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

DESCRIPTION

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

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

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

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

PARAMETERS

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

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_AutoMount, fat_EnumDevice, fat_EnumPartition, fat_MountPartition
void fat_InitUCOSMutex( int mutexPriority );

DESCRIPTION

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

1. The statement #define FAT_USE_UCOS_MUTEX must come before the statement:
   
   #use FAT.LIB

2. After calling OSInit() and before starting any tasks that use the FAT, call
   
   fat_InitUCOSMutex(mutexPriority). The parameter mutexPriority is a µC/OS-II task priority that must be higher than the priorities of all tasks that call FAT API functions.

3. You must not call low-level, non-API FAT or write-back cache functions. Only call FAT functions appended with “fat_” and with public function descriptions.

4. Run the FAT in blocking mode (#define FAT_BLOCK).

Mutex timeouts or other errors will cause a run-time error -ERR_FAT_MUTEX_ERROR. µC/OS-II may raise the priority of tasks using mutexes to prevent priority inversion.

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

PARAMETERS

mutexPriority A µC/OS-II task priority that MUST be higher than the priorities of all tasks that call FAT API functions.

RETURN VALUE

None: success.

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

LIBRARY

FAT.LIB

SEE ALSO

fat_AutoMount, fat_Init
fat_IsClosed

int fat_IsClosed( FATfile far * file);

DESCRIPTION

Returns non-zero if the FATfile passed is not open and zero if open.

(Currently implemented as a macro, but may be modified to be an actual function in a future release.)

PARAMETER

file Pointer to a FATfile structure to check.

RETURN VALUE

1: file is closed
0: file is open

LIBRARY

fat.lib

SEE ALSO

fat_ReadDir, fat_Status, fat_LastAccess, fat_LastWrite
fat_IsOpen

int fat_IsOpen( FATfile far * file);

DESCRIPTION
Returns TRUE if the FATfile passed is in an open state and FALSE otherwise.

(Currently implemented as a macro, but may be modified to be an actual function in a future release.)

PARAMETER
file Pointer to a FATfile structure to check.

RETURN VALUE
!0 if file is open
0 if file is closed

LIBRARY
fat.lib

SEE ALSO
fat_ReadDir, fat_Status, fat_LastAccess, fat_LastWrite
int fat_LastAccess( fat_dirent *entry, struct tm *t );

DESCRIPTION
Puts the last access date of the specified entry into the system time structure \( t \). The time is always set to midnight. The function does not fill in the \texttt{tm}_\texttt{wday} field in the system time structure.

PARAMETERS
- \texttt{entry} Pointer to a directory entry
- \texttt{t} Pointer to a system time structure

RETURN VALUE
- 0: success.
- -\texttt{EINVAL}: invalid directory entry or time pointer

LIBRARY
\texttt{FAT.LIB}

SEE ALSO
fat_ReadDir, fat_Status, fat_CreateTime, fat_LastWrite
int fat_LastWrite( fat_dirent *entry, struct tm *t );

DESCRIPTION

Puts the date and time of the last write for the given entry into the system time structure \( t \). The function does not fill in the \( \text{tm}_\text{wday} \) field in the system time structure.

PARAMETERS

- **entry**: Pointer to a directory entry
- **t**: Pointer to a system time structure

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

- fat_ReadDir,
- fat_Status,
- fat_CreateTime,
- fat_LastAccess
int fat_MountPartition( fat_part *part );

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

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

PARAMETER
part Pointer to the FAT partition structure to mount.

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

LIBRARY
FAT.LIB

SEE ALSO
fat_EnumPartition, fat_UnmountPartition
fat_Open

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

**DESCRIPTION**

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

**PARAMETERS**

- **part**
  Handle for the partition being used.

- **name**
  Pointer to the full path name of the file to be opened/created.

- **type**
  FAT_FILE or FAT_DIR, depending on what is to be opened/created.

- **ff**
  File flags, must be one of:
  - FAT_OPEN - Object must already exist. If it does not exist, -ENOENT will be returned.
  - FAT_CREATE - Object is created only if it does not already exist
  - FAT_MUST_CREATE - Object is created, and it must not already exist.
  - FAT_READONLY - No write operations (this flag is mutually exclusive with any of the CREATE flags).
  - FAT_SEQUENTIAL - Optimize for sequential reads and/or writes. This setting can be changed while the file is open by using the fat_fcntl() function.

- **file**
  Pointer to an empty FAT file structure that will act as a handle for the newly opened file. Note that you must memset this structure to zero when you are using the non-blocking mode before calling this function the first time. Keep calling until something other than -EBUSY is returned, but do not change anything in any of the parameters while doing so.

- **prealloc**
  An initial byte count if the object needs to be created. This number is rounded up to the nearest whole number of clusters greater than or equal to 1. This parameter is only used if one of the *_CREATE flag is set and the object does not already exist. On return, *prealloc is updated to the actual number of bytes allocated. May be NULL, in which case one cluster is allocated if the call is successful.
**RETURN VALUE**

0: success.
-EINVAL: invalid arguments. Trying to create volume label, or conflicting flags.
-ENOENT: file/directory could not be found.
-EINVAL: Invalid path string for parent directory
-EEXIST: object existed when FAT_MUST_CREATE flag set.
-EPERM: trying to create a file/directory on a read-only partition.
-EMFILE - too many open files. If you get this code, increase the FAT_MAXMARKERS definition in the BIOS.

Other negative values indicate I/O error, etc.

Non-blocking mode only:

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

**LIBRARY**

FAT.LIB

**SEE ALSO**

fat_ReadDir, fat_Status, fat_Close
int fat_OpenDir( fat_part *part, char *dirname, FATfile *dir );

DESCRIPTION

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

PARAMETERS

- **part**  
  Pointer to the partition structure being used.

- **dirname**  
  Pointer to the full path name of the directory to be opened or created.

- **dir**  
  Pointer to directory requested.

RETURN VALUE

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

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

LIBRARY

- FAT.LIB

SEE ALSO

fat_ReadDir, fat_Status, fat_Open, fat_Close
fat_PartitionDevice

int fat_PartitionDevice( mbr_dev *dev, int pnum );

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

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

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

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

PARAMETERS
dev Pointer to the device structure of the device to be partitioned.
pnum Partition number of the partition being modified.

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

LIBRARY
FAT.LIB

SEE ALSO
fat_FormatDevice, fat_EnumDevice, fat_FormatPartition
int fat_Read( FATfile *file, char *buf, int len );

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

PARAMETERS

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

RETURN VALUE
Number of bytes read: success. May be less than the requested amount in non-blocking mode, or if EOF was encountered.
-EEOF: starting position for read was at (or beyond) end-of-file.
-EIO: device I/O error.
-EINVAL: file, buf, or len, contain invalid values.
-EPERM: the file is locked.
-ENOENT: the file/directory does not exist.
-EFSTATE: file is in inappropriate state (Only if non-blocking).

LIBRARY
FAT.LIB

SEE ALSO
fat_Open, fat_Write, fat_Seek
**fat_ReadDir**

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

**DESCRIPTION**

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

**PARAMETERS**

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

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

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

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

- **FATATTR_READ_ONLY** - include read-only entries
- **FATATTR_HIDDEN** - include hidden entries
- **FATATTR_SYSTEM** - include system entries
- **FATATTR_VOLUME_ID** - include label entries
- **FATATTR_DIRECTORY** - include directory entries
- **FATATTR_ARCHIVE** - include modified entries
- **FAT_FIL_RD_ONLY** - filter on read-only attribute
- **FAT_FIL_HIDDEN** - filter on hidden attribute
- **FAT_FIL_SYSTEM** - filter on system attribute
- **FAT_FIL_LABEL** - filter on label attribute
- **FAT_FIL_DIR** - filter on directory attribute
- **FAT_FIL_ARCHIVE** - filter on modified attribute

The **FAT_INC_*** flags default to **FAT_INC_ACTIVE** if none set:

- **FAT_INC_DELETED** - include deleted entries
- **FAT_INC_EMPTY** - include empty entries
- **FAT_INC_LNAME** - include long name entries
- **FAT_INC_ACTIVE** - include active entries

The following predefined filters are available:

- **FAT_INC_ALL** - returns ALL entries of ANY type
- **FAT_INC_DEF** - default (files and directories including read-only and archive)

**Note:** Active files are included by default unless **FAT_INC_DELETED**, **FAT_INC_EMPTY**, or **FAT_INC_LNAME** is set. Include flags become the desired filter value if the associated filter flags are set.
EXAMPLES OF FILTER BEHAVIOR

\[\text{mode} = \text{FAT\_INC\_DEF} \mid \text{FAT\_FIL\_HIDDEN} \mid \text{FAT\_ATTR\_HIDDEN}\]

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

\[\text{mode} = \text{FAT\_INC\_DEF} \mid \text{FAT\_FIL\_HIDDEN} \mid \text{FAT\_FIL\_DIR} \mid \text{FAT\_ATTR\_HIDDEN}\]

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

\[\text{mode} = \text{FAT\_INC\_DEF} \mid \text{FAT\_FIL\_HIDDEN} \mid \text{FAT\_FIL\_DIR} \mid \text{FAT\_ATTR\_HIDDEN} \mid \text{~FAT\_ATTR\_DIRECTORY}\]

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

\[\text{mode} = \text{FAT\_INC\_ALL} \mid \text{~FAT\_INC\_EMPTY}\]

would return the next non-empty entry of any type

RETURN VALUE

0: success.

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

Other negative values from the \text{fat\_Open()} call are also possible.

LIBRARY

FAT.LIB

SEE ALSO

fat\_OpenDir, fat\_Status
fat_Seek

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

DESCRIPTION

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

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Pointer to the file structure of the open file.</td>
</tr>
<tr>
<td>pos</td>
<td>Position value in number of bytes (may be negative). This value is interpreted according to the third parameter, whence.</td>
</tr>
<tr>
<td>whence</td>
<td>Must be one of the following:</td>
</tr>
</tbody>
</table>

- **SEEK_SET** - pos is the byte position to seek, where 0 is the first byte of the file. If pos is less than 0, the position pointer is set to 0 and no error code is returned. If pos is greater than the length of the file, the position pointer is set to EOF and error code -EEOF is returned.

- **SEEK_CUR** - seek pos bytes from the current position. If pos is less than 0 the seek is towards the start of the file. If this goes past the start of the file, the position pointer is set to 0 and no error code is returned. If pos is greater than 0 the seek is towards EOF. If this goes past EOF the position pointer is set to EOF and error code -EEOF is returned.

- **SEEK_END** - seek to pos bytes from the end of the file. That is, for a file that is x bytes long, the statement:
  ```c
  fat_Seek (&my_file, -1, SEEK_END);
  ```
  will cause the position pointer to be set at x-1 no matter its value prior to the seek call. If the value of pos would move the position pointer past the start of the file, the position pointer is set to 0 (the start of the file) and no error code is returned. If pos is greater than or equal to 0, the position pointer is set to EOF and error code -EEOF is returned.

- **SEEK_RAW** - is similar to SEEK_SET, but if pos goes beyond EOF, using SEEK_RAW will set the file length and the position pointer to pos.
fat.Seek (cont’d)

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_Open, fat_Read, fat_Write, fat_xWrite
int fat_SetAttr( FATfile *file, int attr );

DESCRIPTION

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

PARAMETERS

file Handle to the open file.
attr Attributes to set in file. For attribute description see fat_GetAttr(). May be one or more of the following:
  • FATATTR_READ_ONLY
  • FATATTR_HIDDEN
  • FATATTR_SYSTEM
  • FATATTR_VOLUME_ID
  • FATATTR_DIRECTORY
  • FATATTR_ARCHIVE
  • FATATTR_LONG_NAME

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_Open, fat_Status
**fat_Split**

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

**DESCRIPTION**

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

**PARAMETERS**

- **file**
  - Pointer to the open file to split.
- **where**
  - May be one of the following:
    - $\geq 0$ - absolute byte to split the file. If the absolute byte is beyond the EOF, file is split at EOF.
    - `FAT_BRK_END` - split at EOF.
    - `FAT_BRK_POS` - split at current file position.
- **newfile**
  - Pointer to the absolute path and name of the new file created for the split.

**RETURN VALUE**

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

**LIBRARY**

FAT.LIB

**SEE ALSO**

fat_Open, fat_OpenDir, fat_Delete, fat_Truncate, fat_Close
fat_Status

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

DESCRIPTION

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

PARAMETERS

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

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_ReadDir
int fat_SyncFile( FATfile *file );

DESCRIPTION

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

PARAMETERS

file Pointer to the open file.

RETURN VALUE

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

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

LIBRARY

FAT.LIB

SEE ALSO

fat_Close, fat_Open, fat_OpenDir
**fat_SyncPartition**

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

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

**PARAMETER**
- `part` Pointer to the partition to be synchronized.

**RETURN VALUE**
- 0: success.
-EINVAL: `part` is invalid.
-EBUSY: the device is busy (Only if non-blocking). Call function again to complete the sync.
Any other negative value: I/O error when updating the device.

**LIBRARY**
FAT.LIB

**SEE ALSO**
- `fat_Close`, `fat_SyncFile`, `fat_UnmountPartition`
int fat_Tell( FATfile *file, unsigned long *pos );

DESCRIPTION

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

µC/OS-II USERS:

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

    #define FAT_USE_UCOS_MUTEX

• Mutex timeouts or other mutex errors will cause the run-time error
  ERR_FAT_MUTEX_ERROR. The default mutex timeout is 5 seconds and can be
  changed by #define’ing a different value for FAT_MUTEX_TIMEOUT_SEC.

• You MUST call fat_InitUCOSMutex() after calling OSInit() and before calling
  any other FAT API functions.

• You must run the FAT in blocking mode (#define FAT_BLOCK).

• You must not call low-level, non-API FAT or write-back cache functions. Only call FAT
  functions appended with “fat_” and with public function descriptions.

PARAMETERS

  file Pointer to the file structure of the open file

  pos Pointer to the variable where the value of the file position pointer is to be
        placed.

RETURN VALUE

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

LIBRARY

  FAT.LIB

SEE ALSO

  fat_Seek, fat_Read, fat_Write, fat_xWrite
int fat_tick( void );

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

RETURN VALUE
Currently always 0.

LIBRARY
FATWTC.LIB
int fat_Truncate( FATfile *file, long where );

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

PARAMETERS
- **file**: Pointer to the open file to truncate.
- **where**: One of the following:
  - \( \geq 0 \) - absolute byte to truncate the file. The file is truncated at EOF if 
    the absolute byte is beyond EOF.
  - FAT_BRK_END - truncate at EOF.
  - FAT_BRK_POS - truncate at current file position.

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

LIBRARY
FAT.LIB

SEE ALSO
fat_Open, fat_OpenDir, fat_Delete, fat_Split
`fat_UnmountDevice`  

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

**DESCRIPTION**

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

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

**PARAMETER**

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

**RETURN VALUE**

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

**LIBRARY**

FAT.LIB

**SEE ALSO**

`fat_NumDevice`, `fat_AutoMount`, `fat_UnmountPartition`
int fat_UnmountPartition( fat_part *part );

DESCRIPTION

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

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

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

PARAMETERS

part Pointer to a FAT partition structure to unmount.

RETURN VALUE

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

LIBRARY

FAT.LIB

SEE ALSO

fat_enumPartition, fat_MountPartition, fat_UnmountDevice
fat_Write

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

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

PARAMETERS
 file Handle for the open file being written.
 buf Pointer to the buffer containing data to write.
 len Length of data to be written.

RETURN VALUE
 Number of bytes written: success (may be less than len, or zero if non-blocking mode)
 -EIO: device I/O error.
 -EINVAL: file, buf, or len contain invalid values.
 -ENOENT: file does not exist.
 -ENOSPC: no space left on the device to complete the write.
 -EFAULT: problem in file (broken cluster chain, etc.).
 -EPERM: the file is locked or is write-protected.
 -EBUSY: the device is busy (only if non-blocking).
 -EFSTATE: file is in inappropriate state (only if non-blocking).

LIBRARY
 FAT.LIB

SEE ALSO
 fat_Open, fat_Read, fat_xWrite, fat_Seek
**fat_xRead**

fat_xRead( FATfile * file, char far * buf, int len );

**DESCRIPTION**

Given `file`, `buf` and `len`, this routine reads `len` characters from the specified file and places the characters into string `buf`. Returns the number of characters actually read on success.

Characters will be read beginning at the current position of the file and the position pointer will be left pointing to the next byte to be read. The file position can be manually set with the `fat_Seek()` function. If the file contains less the “`len`” characters from the current position to the end of the file (EOF), then the transfer will stop at the EOF. If already at the EOF, -EEOF is returned. The `len` parameter must be positive, limiting reads to 32767 bytes per call.

**µC/OS-II USERS:**

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

- Mutex timeouts or other mutex errors cause a run-time error ERR_FAT_MUTEX_ERROR. The default mutex timeout is 5 seconds and can be changed by #define'ing a different value for FAT_MUTEX_TIMEOUT_SEC.

- You MUST call `fat_InitUCOSMutex()` after calling `OSInit()` and before calling any other FAT API functions.

- You must run the FAT in blocking mode (#define FAT_BLOCK).

- You must not call low-level, non-API FAT or write-back cache functions. Only call FAT functions appended with “fat_” and with public function descriptions.

**PARAMETERS**

- **file**
  Handle for the file being read

- **buf**
  Pointer to buffer where data is to be placed. May be NULL in order to discard data

- **len**
  Length of data to be read. If this is zero, then the return code will be ‘1’ if not at EOF, or ‘0’ if at EOF.
fat_xRead (cont’d)

RETURN VALUE
    Number of bytes read on Success. May be less than the requested amount in non-blocking mode, or if EOF was encountered.
    - EEOF: stating position for read was at (or beyond) EOF.
    - EIO: on device IO error
    - EINVAL: if file, buf, or len contain invalid values
    - EPERM: if the file is locked
    - ENOENT: if file/directory does not exist
    - EFSTATE: if file in inappropriate state (non-blocking)

SEE ALSO
    fat_Open, fat_Read, fat_Write, fat_xWrite, fat.Seek
Fat_xWrite

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

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

Parameters
- file: Handle for the open file being written.
- xbuf: xmem address of the buffer to be written.
- len: Length of data to write.

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

Library
FAT.LIB

See Also
fat_Open, fat_Read, fat_Write, fat_Seed
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**

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

```c
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)**

```c
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)
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
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)
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)
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 x.

blockexp Pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplxinv, fftreal, fftrealinv, hancplx, hannreal, powerspectrum
fftcplxinv

```c
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 \( x \).
- **blockexp**: Pointer to integer block exponent.

**LIBRARY**

FFT.LIB

**SEE ALSO**

`fftcplx`, `fftreal`, `fftrealinv`, `hanncplx`, `hannreal`, `powerspectrum`
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 \( \text{blockexp} \) 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
\( \text{blockexp} \) Pointer to integer block exponent.

LIBRARY
FFT.LIB

SEE ALSO
fftclpx, fftclpxinv, fftrealinv, hannclpx, hannreal, powerspectrum
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 \(\text{blockexp}\) is increased by 1 each time array \(x\) has to be scaled, to avoid arithmetic overflow. The value of \(\text{blockexp}\) is also decreased by \(\log_2N\) 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 \(x\).
- \(\text{blockexp}\) Pointer to integer block exponent.

LIBRARY
FFT.LIB

SEE ALSO
- fftcplx, fftcplxinv, fftreal, hanncplx, hannreal, powerspectrum
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_erasesector, flash_gettype, flash_init, flash_read, flash_readsector, flash_sector2xwindow, flash_writesector
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
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

```c
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_erasesector, flash_gettype, flash_read, flash_readsector, flash_sector2xwindow, flash_writesector`
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 FlashDescriptor 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 paddr.

\( length \) The number of bytes to read.

RETURN VALUE
0: Success.

LIBRARY
FLASH.LIB

SEE ALSO
flash_erasechip, flash_erasesector, flash_gettype, flash_init, flash_readsector, flash_sector2xwindow, flash_writesector, paddr
**flash_readsector**

```c
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 FlashDescriptor 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 `paddr()`.

**RETURN VALUE**

0: Success.

**LIBRARY**

FLASH.LIB

**SEE ALSO**

`flash_erasechip`, `flash_erasesector`, `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_erasesector, flash_gettype, flash_init, flash_read, flash_readsector, 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 FlashDescriptor 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 paddr().

RETURN VALUE

0: Success.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_gettype, flash_init, flash_read, flash_readsector, flash_sector2xwindow
### floor

```c
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

```c
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 × 10.3 = 2.1.

**LIBRARY**
MATH.LIB

**SEE ALSO**
`ceil`, `floor`
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)

```c
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)`
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
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**

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

**DESCRIPTION**

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

**LIBRARY**

SYS.LIB

---

**fread (FS1)**

```c
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


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 fopen_rd(), fopen_wr() or fcreate()).
buf Data buffer located in root data memory or stack. This must be dimensioned with at least len 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, \( f \times ( 2^n ) \).

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
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.

```c
#define FS_FLASH
#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
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)**

```c
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 µC/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)**

```c
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)**

```c
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)

```c
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)**

```c
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 fopen_rd(), fopen_wr() or fcreate()).

- **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**

fteell (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 lx, 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

   lx  Logical extent number to query.

   all Boolean: 0 for current free capacity only, 1 for total. Must use 1 if calling before fs_init().

   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
Dynamic C Functions

---

**fs_get_other_lx (FS2)**

```c
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 FS_PARTITION_FRACTION or FS_MODIFY_EXTENT. The following parameters are ignored if this parameter is not FS_PARTITION_FRACTION.
**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)*
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)`
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
long ftell( File * f );

DESCRIPTION
Return the current read/write position of the file. Bytes in a file are sequentially numbered starting at zero. If the current position is zero, then the first byte of the file will be read or written. If the position equals the file length, then no data can be read, but any write will append data to the file.

Note that no checking is done to see if the file descriptor is valid. If the File is not actually open, the return value will be random.

PARAMETERS
f Pointer to file descriptor (initialized by fopen_rd(), fopen_wr() or fcreate()).

RETURN VALUE
Current read/write position (0 to length-of-file).

ERRNO VALUES
None

LIBRARY
fs2.lib

SEE ALSO
fseek (FS2)
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 fopen_wr() or fcreate()).
- **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 len bytes. This parameter may also be null if the deleted data is not needed.
fshift (cont’d)

RETURN VALUE

len: Success.
<len: Partial success - returns amount successfully deleted. errno gives further details (probably ENOSPC)
0: Error or len was zero.

ERRNO VALUES

EBADFD - File descriptor not opened, or is read-only.
EINVAL - len less than zero.
0 - Success, but len was zero.
EIO - I/O error.
ENOSPC - extent out of space.
EBUSY - file opened more than once. This is only possible if
FS2_SHIFT_DOESNT_UPDATE_FPOS is not defined, which is the default case.

LIBRARY

FS2.LIB

SEE ALSO

fread (FS2), fwrite (FS2)
fwrite (FS1)

int fwrite( File * f, char * buf, int len );

DESCRIPTION
Appends len bytes from the source buffer to the end of the file.

PARAMETERS
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 fopen_wr() or fcreate()).
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)
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: \( \text{ftoa}(1.0, \text{buf}) \) yields \( \text{buf} = "100000000" \) and returns \(-8\).

PARAMETERS
- \( f \) : Float number to convert.
- \( \text{buf} \) : Converted string. The string is no longer than 10 characters long.

RETURN VALUE
The exponent of the number.

LIBRARY
\texttt{STDIO.LIB}

SEE ALSO
- \texttt{utoa, itoa}

---

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
\texttt{STDIO.LIB}

SEE ALSO
- \texttt{gets, putchar}
get_cpu_frequency

unsigned long get_cpu_frequency();

DESCRIPTION
Returns the clock speed of the CPU as calculated by the BIOS, adjusted for the clock doubler if it is enabled. Due to the limited precision of the clock speed calculation, the calculated and actual clock speeds may differ slightly.

RETURN VALUE
The clock speed of the CPU in Hz.

LIBRARY
sys.lib

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
char getdivider19200( void );

DESCRIPTION

This function returns a value that is used in baud rate calculations. The correct value is returned regardless of the compile mode. In separate I&D space mode, the divider value is stored as a define byte in code space, so directly accessing the variable will result in an incorrect load (from constant data space). This function uses the `ldp` instruction, which circumvents the separate I&D default loading scheme so that the correct value is returned.

RETURN VALUE

The value used in baud rate calculation.

LIBRARY

SYS.LIB

gets

char * gets( char * s );

DESCRIPTION

Waits for a string terminated by <CR> at the stdio window. The string returned is null terminated without the return. The user should make sure only one process calls this function at a time.

PARAMETERS

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
**_GetSysMacroIndex**

```c
int _GetSysMacroIndex( int n, char * buf, uint32 * value );
```

**DESCRIPTION**

Skips to the nth macro entry and retrieves the macro name (as defined by the compiler), and the value of the macro as defined in the system macro table. The system macro table contains board specific configuration parameters that are defined by the compiler and can be retrieved at runtime through this interface. The flash driver must be initialized and the System ID block must be read before this function will return accurate results.

This function only applies to boards with Version 5 or later System ID blocks.

**PARAMETERS**

- `n`  
The index in the system macro table.

- `buf`  
Character array to contain and return macro name (copied from system macro table). MUST BE AT LEAST SYS_MACRO_LENGTH bytes or function will overflow buffer and can crash system!

- `value`  
Pointer to macro value to return to caller.

**RETURN VALUE**

- `0`: if successful
- `-1`: invalid address or range (use to find end of table)
- `-2`: ID block or macro table invalid

**LIBRARY**

- IDBLOCK.LIB

**SEE ALSO**

- _GetSysMacroValue


```c
int _GetSysMacroValue( char * name, long * value );
```

**DESCRIPTION**

Finds the system table macro named by the first parameter (as defined by the compiler) and retrieves the value of the macro as defined in the system macro table. The system macro table contains board specific configuration parameters that are define by the compiler and can be retrieved at runtime through this interface. The flash driver must be initialized and the System ID block must be read before this function will return accurate results.

See `writeUserBlockArray` for more details.

This function only applies to boards with Version 5 or later System ID blocks.

**PARAMETERS**

- `name` Name of System ID block macro (acts as lookup key).
- `value` Pointer to macro value to return to caller.

**RETURN VALUE**

- 0: if successful
- -1: Macro name not found
- -2: No valid ID block found (block version 3 or later)
- -3: First parameter is a bad macro name

**LIBRARY**

`IDBLOCK.LIB`

**SEE ALSO**

`writeUserBlockArray`
unsigned GetVectExtern2000( void );

DESCRIPTION

Reads the address of external interrupt table entry. This function really just returns what is present
in the table. The return value is meaningless if the address of the external interrupt has not been
written.

This function should be used for Rabbit 2000 processors that are marked IQ2T in the 3rd line
of text across the face of the chip. It will work for other versions of the Rabbit 2000 but should
be deprecated in favor of GetVectExtern3000() which allows the use of two external
interrupts. (Please see document TN301, “Rabbit 2000 Microprocessor Interrupt Issue,” on the
Rabbit Semiconductor website for more information.)

RETURN VALUE

Jump address in vector table.

LIBRARY

SYS.LIB

SEE ALSO

GetVectIntern, SetVectExtern2000, SetVectIntern,
GetVectExtern3000
unsigned GetVectExtern3000( int interruptNum );

DESCRIPTION

Reads the address of an external interrupt table entry. This function may be used with all Rabbit 3000 processors and all Rabbit 2000 processors with the exception of the ones marked IQ2T in the 3rd line of text across the face of the chip. For those, use the function GetVectExtern2000() instead.

GetVectExtern3000() returns the value at address:

(external vector table base) + (interruptNum * 8) + 1

PARAMETER

interruptNum  Interrupt number. Should be 0 or 1.

RETURN VALUE

Jump address in vector table.

LIBRARY

SYS.LIB

SEE ALSO

SetVectExtern3000, SetVectIntern, GetVectIntern, GetVectExtern2000
GetVectIntern

unsigned GetVectIntern( int vectNum );

DESCRIPTION

Reads the address of the internal interrupt table entry and returns whatever value is at the address:

(internal vector table base) + (vectNum*16) + 1

PARAMETER

vectNum            Interrupt number; should be 0–15.

RETURN VALUE

Jump address in vector table.

LIBRARY

SYS.LIB

SEE ALSO

GetVectExtern2000, SetVectExtern2000, SetVectIntern

gps_get_position

int gps_get_position( 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 GPSPositon 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
void hanncplx(int * x, int N, int * blockexp);

DESCRIPTION

Convolves 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
void hannreal( int * x, int N, int * blockexp );

DESCRIPTION
Convolves an \( N \)-point positive-frequency complex spectrum with the three-point Hann kernel. The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero–crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

The imaginary part of the dc term (stored in \( x[1] \)) is considered to be the real part of the \( f_{max} \) term. The dc and \( f_{max} \) spectral components take part in the convolution along with the other spectral components. The real part of \( f_{max} \) component affects the real part of the \( X[N-1] \) component (and vice versa), and should not arbitrarily be set to zero unless these components are unimportant.

PARAMETERS
\( x \) Pointer to \( N \)-element array of complex fractions.
\( N \) Number of complex elements in array \( x \).
\( \text{blockexp} \) Pointer to integer block exponent.

RETURN VALUE
None. The filtered spectrum replaces the original spectrum.

LIBRARY
FFT.LIB

SEE ALSO
fftclpx, fftclpxinv, fftreal, fftrealinv, hannclpx, powerspectrum
HDLCabortX

```c
void HDLCabortX( void ); /* Where X is E or F */
```

**DESCRIPTION**

Immediately stops any transmission. An HDLC abort code will be sent if the driver was in the middle of sending a packet.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

**LIBRARY**

HDLC_PACKET.LIB

---

HDLCcloseX

```c
void HDLCcloseX( void ); /* Where X is E or F */
```

**DESCRIPTION**

Disables the HDLC port (E or F). If it was used, the TAT1R resource (timer A1 cascade) is released. This function is non-reentrant.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

**LIBRARY**

HDLC_PACKET.LIB

**SEE ALSO**

TAT1R_SetValue
HDLCdropX

int HDLCdropX( void ); /* Where X is E or F */

DESCRIPTION

Drops the next received packet, freeing up its buffer. This must be used if the packet has been
examined with HDLCpeekX() and is no longer needed. A call to HDLCreviceX() is the
only other way to free up the buffer.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

RETURN VALUE

1: Packet dropped.
0: No received packets were available.

LIBRARY

HDLC_PACKET.LIB

HDLCerrorX

int HDLCerrorX( unsigned long * bufptr, int * lenptr );
/* Where X is E or F */

DESCRIPTION

This function returns a set of possible error flags as an integer. A received packet with errors is
automatically dropped.

Masks are used to check which errors have occurred. The masks are:
• HDLC_NOBUFFER - driver ran out of buffers for received packets.
• HDLC_OVERRUN - a byte was overwritten and lost before the ISR could retreive it.
• HDLC_OVERFLOW - a received packet was too long for the buffers.
• HDLC_ABORTED - a received packet was aborted by the sender during tranmission.
• HDLC_BADCRC - a packet with an incorrect CRC was received.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

RETURN VALUE

Error flags (see above).

LIBRARY

HDLC_PACKET.LIB
void HDLCextClockE( int ext_clock ) /* Where X is E or F */

DESCRIPTION

Configures HDLC to be either internally (default) or externally clocked. This should be called after HDLCopenX().

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

PARAMETER

ext_clock 1 for externally clocked
0 for internally clocked

LIBRARY

HDLC_PACKET.LIB
int HDLCopenX( long baud, char encoding, unsigned long buffers, int buffer_count, int buffer_size ); /* Where X is E or F */

DESCRIPTION
Opens serial port E or F in HDLC mode. Sets up buffers to hold received packets. This function is intended for use with the Rabbit 3000 and Rabbit 4000. Please see the chip manuals for more details on HDLC and the bit encoding modes to use.

PARAMETERS
- **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:
  - 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:
  \[(\text{# of buffers}) \times ((\text{size of buffer}) + 4)\]
- **buffer_count**: The number of buffers in the block pointed to by `buffer`.
- **buffer_size**: The capacity of each buffer in the block pointed to by `buffer`.

RETURN VALUE
- 1: Actual baud rate is within 5% of the requested baud rate,
- 0: Otherwise.

LIBRARY
HDLC_PACKET.LIB

SEE ALSO
SetSerialTATxRValues, TAT1R_SetValue
HDLCpeekX

```c
int HDLCpeekX( unsigned long * bufptr, int * lenptr );
/* Where X is E or F */
```

**DESCRIPTION**

Reports the location and size of the next available received packet if one is available. This function can be used to efficiently inspect a received packet without actually copying it into a root memory buffer. Once inspected, the buffer can be received normally (see HDLCreceiveX()), or dropped (see HDLCdropX()).

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

**PARAMETERS**

- **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**

HDLC_PACKET.LIB
int HDLCreceiveX(char *rx_buffer, int length); /* Where X is E or F */

DESCRIPTION
Copies a received packet into rx_buffer if there is one. Packets are received in the order they arrive, even if multiple packets are currently stored in buffers.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

PARAMETERS
rx_buffer Pointer to the buffer to copy a received packet into.
length Size of the buffer pointed to by rx_buffer.

RETURN VALUE
≥0: Size of received packet.
-1: No packets are available to receive.
-2: The buffer is not large enough for the received packet. In this case, the packet remains in the receive buffer.

LIBRARY
HDLC_PACKET.LIB
HDLCsendX

```c
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 and Rabbit 4000.

**PARAMETERS**
- **tx_buffer**  A pointer to the packet to be sent. This buffer must not change while transmitting (see above.)
- **length**  The size of the buffer (in bytes).

**RETURN VALUE**
- 1: Sending packet.
- 0: Cannot send, another packet is currently being transmitted.

**LIBRARY**
HDLC_PACKET.LIB
**HDLCsendingX**

```c
define HDLCsendingX( void ); /* Where X is E or F */
```

**DESCRIPTION**

Returns true if a packet is currently being transmitted. This function is intended for use with the Rabbit 3000 and Rabbit 4000.

**RETURN VALUE**

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

**LIBRARY**

HDLC_PACKET.LIB

---

**hexstrtobyte**

```c
define hexstrtobyte ( char far *p );
```

**DESCRIPTION**

Converts two hex characters (0-9A-Fa-f) to a byte.

**RETURN VALUE**

The byte (0-255) represented by the two hex characters or -1 on error (invalid character, string less than 2 bytes).

**EXAMPLES**

hexstrtobyte("FF") returns 255
hexstrtobyte("0") returns -1 (error because < 2 characters)
hexstrtobyte("ABCDEF") returns 0xAB (ignores additional chars)
**hitwd**

```c
void hitwd( void );
```

**DESCRIPTION**

Hits the watchdog timer, postponing a hardware reset for 2 seconds. Unless the watchdog timer is disabled, a program must call this function periodically, or the controller will automatically reset itself. If the virtual driver is enabled (which it is by default), it will call hitwd in the background. The virtual driver also makes additional “virtual” watchdog timers available.

**LIBRARY**

VDRIVER.LIB

---

**htoa**

```c
char * htoa( 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

```c
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

```c
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**

```c
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**

```c
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

```c
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

```c
int iscntrl( int c );
```

**DESCRIPTION**
Tests for a control character: 0 <= c <= 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
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**

```c
int isgraph(int c);
```

**DESCRIPTION**
Tests for a printing character other than a space: 33 ≤ c ≤ 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**

```c
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**

```c
int isspace( int c );
```

**DESCRIPTION**
Tests for a white space, character, tab, return, newline, vertical tab, form feed, and space:
9 <= c <= 13 and c == 32.

**PARAMETERS**

- `c` Character to test.

**RETURN VALUE**

- 0 if not, !0 otherwise.

**LIBRARY**

STRING.LIB

**SEE ALSO**

- ispunct

---

**isprint**

```c
int isprint( int c );
```

**DESCRIPTION**
Tests for printing character, including space: 32 <= c <= 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

```c
int ispunct( int c );
```

**DESCRIPTION**

Tests for a punctuation character.

<table>
<thead>
<tr>
<th>Character</th>
<th>Decimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>32</td>
</tr>
<tr>
<td>! &quot; # $ % &amp; ' ( ) * + , - . /</td>
<td>33 &lt;= c &lt;= 47</td>
</tr>
<tr>
<td>; ; &lt; = &gt; ? @</td>
<td>58 &lt;= c &lt;= 64</td>
</tr>
<tr>
<td>[ ] ^ _ \ `</td>
<td>91 &lt;= c &lt;= 96</td>
</tr>
<tr>
<td>{ } ~</td>
<td>123 &lt;= c &lt;= 126</td>
</tr>
</tbody>
</table>

**PARAMETERS**

- `c` Character to test.

**RETURN VALUE**

- 0: Not a character.
- ! 0: Is a character.

**LIBRARY**

- STRING.LIB

**SEE ALSO**

- isspace
**isupper**

```c
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**

```c
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
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

```c
int i2c_check_ack( void );
```

**DESCRIPTION**

Checks if slave pulls data low for ACK on clock pulse. Allows for clocks stretching on SCL going high.

**RETURN VALUE**

- 0: ACK sent from slave.
- 1: NAK sent from slave.
- ~1: Timeout occurred.

**LIBRARY**

I2C.LIB

**SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*. 


**i2c_init**

```c
void i2c_init( void );
```

**DESCRIPTION**

Sets up the SCL and SDA port pins for open-drain output.

**LIBRARY**

I2C.LIB

**SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

**i2c_read_char**

```c
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

```c
int i2c_send_ack( void );
```

**DESCRIPTION**

Sends ACK sequence to slave. ACK is usually sent after a successful transfer, where more bytes are going to be read.

**RETURN VALUE**

- 0: Success.
- -1: Clock stretching timeout.

**LIBRARY**

I2C.LIB

**SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

---

### i2c_send_nak

```c
int i2c_send_nak( void );
```

**DESCRIPTION**

Sends NAK sequence to slave. NAK is often sent when the transfer is finished.

**RETURN VALUE**

- 0: Success.
- -1: Clock stretching timeout.

**LIBRARY**

I2C.LIB

**SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.
**i2c_start_tx**

```c
int i2c_start_tx( void );
```

**DESCRIPTION**

Initiates I^2^C transmission by sending the start sequence, which is defined as a high to low transition on SDA while SCL is high. The point being that SDA is supposed to remain stable while SCL is high. If it does not, then that indicates a start (S) or stop (P) condition. This function first waits for possible clock stretching, which is when a bus peripheral holds SCK low.

**RETURN VALUE**

- 0: Success.
- -1: Clock stretching timeout.

**LIBRARY**

`I2C.LIB`

**SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*
int i2c_startw_tx( void );

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

```c
void i2c_stop_tx( void );
```

**DESCRIPTION**

Sends the stop sequence to the slave, which is defined as bringing SDA high while SCL is high, i.e., the clock goes high, then data goes high.

**LIBRARY**

I2C.LIB

**SEE ALSO**

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor*.

### i2c_write_char

```c
int i2c_write_char( char d );
```

**DESCRIPTION**

Sends 8 bits to slave. Checks if slave pull 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**

```c
int kbhit( void );
```

**DESCRIPTION**

Detects keystrokes in the Dynamic C Stdio window.

**RETURN VALUE**

! 0 if a key has been pressed, 0 otherwise.

**LIBRARY**

UTIL.LIB

---

**labs**

```c
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 \( x \times (2^n) \).

PARAMETERS
   x     The value between 0.5 inclusive, and 1.0
   n     An integer

RETURN VALUE
   The result of \( x \times (2^n) \).

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 –INF and signals a domain error when \( x \leq 0 \).

LIBRARY
   MATH.LIB

SEE ALSO
   exp, log10
**log_clean**

```c
int log_clean( LogDest ld );
```

**DESCRIPTION**

Reset only the specified destination class and stream (encoded as a LogDest value). This is only applicable to filesystem or XMEM destinations since they are locally persistent storage. XMEM is automatically cleaned at start-up time, since it is not assumed to be non-volatile.

If this operation is not applicable, 0 is returned with no further action.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

**PARAMETER**

`ld` Destination class and stream. Use one of the constants `LOG_DEST_FS2` or `LOG_DEST_XMEM`, then OR in the stream number (0-63).

**RETURN VALUE**

0: success  
-2: The stream is out-of-range for the class.

**LIBRARY**

`log.lib`
int log_close( LogDestClass ldc );

DESCRIPTION

Close the specified class, enumerating all streams. If the destination class is already closed, returns success.

Note: Please see the comments at the top of log.lib for a description of the message logging subsystem.

PARAMETER

ldc Destination class. Use one of the constants LOG_DEST_FS2, LOG_DEST_XMEM, LOG_DEST_UDP or LOG_DEST_ALL. The latter case closes all open destinations.

RETURN VALUE

0: success

LIBRARY

log.lib
int log_condition( LogDest ldst );

DESCRIPTION

Return the state of the specified log destination. Destination classes or streams that are not configured cause a -2 return code.

Note: Please see the comments at the top of log.lib for a description of the message logging subsystem.

PARAMETER

ldst Destination class and stream. Use one of the constants LOG_DEST_FS2 or LOG_DEST_XMEM, then OR in the stream number (0-63).

RETURN VALUE

0: Destination not open
1: destination OK
2: destination reached limit of its space quota
-1: error in destination.
-2: destination not configured

LIBRARY

log.lib
char * log_format( LogEntry *le, char *buffer, int length, int pfx );

DESCRIPTION

Given the log entry returned by log_next() or log_prev(), format the entry as an ASCII string. The string is constructed in Unix "syslog" format:

```%d>%s %.15s %.8s[%d]: %s```

where the substitutions are:

- `%d`: facility/priority as decimal number (0-255)
- `%.15s`: date/time as "Mon dd hh:mm:ss"
- `%.8s`: process name - taken from LOG_UDP_PNAME(0) if defined, else "" (empty).
- `%s`: process ID, but the entry serial number is used instead.
- `%s`: the log entry data.

A null terminator is always added at buffer[length-1], or at the end of the string if it fits in the buffer. If pfx is zero, then the above syslog prefix is not generated.

Note: Please see the comments at the top of log.lib for a description of the message logging subsystem.

PARAMETERS

- `le`: Log entry result from log_next/log_prev().
- `buffer`: Storage for result. Must be dimensioned at least 'length'.
- `length`: Length of buffer. For the maximum sized log entry, the buffer should be 158 bytes. The minimum length must be greater than or equal to 43 (if pfx true) else 1. If a bad length is passed, the function returns without writing to buffer.
- `pfx`: 0: message text only; do not generate syslog prefix.
  1: prefix plus message text.
  2: prefix only (up to ']', then null terminator).

RETURN VALUE

`buffer` address, or NULL if bad length passed.

LIBRARY

log.lib

SEE ALSO

log_next, log_prev
**log_map**

```c
uint32 log_map( LogFacPri lfp );
```

**DESCRIPTION**

Return the log destination class and stream, for a given facility/priority code. The result is up to four destinations packed into a longword. This function merely invokes the macro `LOG_MAP()`, which may be overridden by the application, but defaults to just the filesystem.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

**PARAMETER**

- **lfp**
  - Facility/priority code. This is a single-byte code specified whenever any log message is added. Facility is coded in the 5 MSBs, and priority in the 3 LSBs.

**RETURN VALUE**

Up to four destinations for a message of the specified facility and priority. Each byte in the resulting long word represents a destination/stream. A zero byte indicates no destination. If the result is all zeros, then a message of this type would be discarded.

**LIBRARY**

- `log.lib`
**log_next**

```c
int log_next( LogDest ldst, LogEntry * le );
```

**DESCRIPTION**
Retrieve next log entry. You must call `log_seek()` before calling this function the first time. Retrieval of stored log messages proceeds, for example, as follows:

```c
log_seek(ldst, 0); // seek to start
log_next(ldst, &L); // get 1st entry
log_next(ldst, &L); // get 2nd entry
log_prev(ldst, &L); // get 2nd entry again
log_prev(ldst, &L); // get 1st entry
log_prev(ldst, &L); // returns -1
```

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

**PARAMETERS**

- **ldst**
  Destination class and stream. Use one of the constants `LOG_DEST_FS2` or `LOG_DEST_XMEM`, then OR in the stream number (0-63).

- **le**
  Storage for result.

**RETURN VALUE**
- non-negative: length of log entry data
- -1: End of log or not open
- -2: Not a readable log destination class

**LIBRARY**
`log.lib`

**SEE ALSO**
`log_seek`, `log_prev`
**log_open**

```c
int log_open( LogDestClass ldc, int clean );
```

**DESCRIPTION**

Open the specified logging destination class. If necessary, this enumerates all possible streams within the class, opening them all (necessary only for FS2 class, since each file needs to be opened). Class `LOG_DEST_ALL` opens all configured classes.

If `clean` is true, then the dest is set to empty log, if that makes sense for the class.

**Note:** Please see the comments at the top of `log.lib` for a description of the message logging subsystem.

**PARAMETERS**

- **ldc**
  
  Destination class: `LOG_DEST_FS2`, `LOG_DEST_UDP`, `LOG_DEST_XMEM` or `LOG_DEST_ALL`.

- **clean**
  
  Boolean, should the destination be erased before using?

**RETURN VALUE**

- `0`: success
- `-1`: unknown LogDestClass value

**LIBRARY**

`log.lib`
int log_prev( LogDest ldst, LogEntry * le );

DESCRIPTION
Retrieve previous log entry. You must call log_seek() before calling this function the first time. Retrieval of stored log messages proceeds, for example, as follows:

log_seek(ldst, 1);  // seek to end
log_prev(ldst, &L);  // get last entry
log_prev(ldst, &L);  // get 2nd last entry
log_next(ldst, &L);  // get 2nd last entry again
log_next(ldst, &L);  // get last entry
log_next(ldst, &L);  // returns -1

Note: Please see the comments at the top of log.lib for a description of the message logging subsystem.

PARAMETERS
ldst Destination class and stream. Use one of the constants LOG_DEST_FS2 or LOG_DEST_XMEM, then OR in the stream number (0-63).

le Storage for result.

RETURN VALUE
- non-negative = length of log entry data
- -1 = Start of log or not open
- -2 = Not a readable log destination class

LIBRARY
log.lib

SEE ALSO
log_seek, log_next
int log_put( LogFacPri ifp, uint8 fmt, const char *data, int length );

DESCRIPTION

Add a log entry. The specified facility/priority is mapped to the appropriate destination(s), as configured by the macros. If the destination exists, then the log entry is added; otherwise, the entry is quietly ignored. If a destination is unable to fit the log entry, and the destination is configured as “circular,” then the first few entries may be deleted to make room. If this cannot be done, or an unrecoverable error occurs, then -2 is returned. For non-circular destinations, -2 is returned when it becomes full.

Since multiple log destinations can result from the given facility/priority, it can be difficult to determine which actual destination caused an error. You can use the log_map() function to determine the destinations, then check each destination's state using log_condition().

Note: Please see the comments at the top of log.lib for a description of the message logging subsystem.

PARAMETERS

ifp Facility/priority code. Facility in 5 MSBs, priority in 3 LSBs.
fmt Format code. 0 for ascii string, others user-defined.
data Pointer to first byte of data to store.
length Length of data. Must be between 0 and 115 (LOG_MAX_MESSAGE) inclusive.

RETURN VALUE

0 = success
-1 = Message too long (over 115).
-2 = Unrecoverable error in destination. This return code usually means that the destination is unusable and further entries for that destination will probably meet the same fate. This can also mean that the destination has not been opened.

LIBRARY

log.lib
int log_seek( LogDest ldst, int whence );

DESCRIPTION
Position log for readback. The next call to log_next() will return the first entry in the log (if whence=0), or log_prev() will return the last entry (if whence=1).

Note: Please see the comments at the top of log.lib for a description of the message logging subsystem.

PARAMETERS
ldst Destination class and stream. Use one of the constants LOG_DEST_FS2 or LOG_DEST_XMEM, then OR in the stream number (0-63).

whence 0: first entry.
1: last entry.
other values reserved.

RETURN VALUE
0 = success.
-1 = Log empty.
-2 = Unrecoverable error or not open.
-3 = Not a seekable or configured log destination class.
-4 = invalid whence parameter.

LIBRARY
log.lib

SEE ALSO
log_next, log_prev
log10

float log10( float x );

DESCRIPTION
Computes the base 10 logarithm of real float value x.

PARAMETERS
x Value to compute

RETURN VALUE
The log base 10 of x.
The function returns –INF and signals a domain error when x ≤ 0.

LIBRARY
MATH.LIB

SEE ALSO
log, exp

longjmp

void longjmp( jmp_buf env, int val );

DESCRIPTION
Restores the stack environment saved in array env[]. See the description of setjmp() for details of use.

Note: you cannot use longjmp() to move out of slice statements, costatements, or cofunctions.

PARAMETERS
env Environment previously saved with setjmp().
val Integer result of setjmp().

LIBRARY
SYS.LIB

SEE ALSO
setjmp
void loophead( void );

DESCRIPTION
This function should be called within the main loop in a program. It is necessary for proper single-user cofunction abandonment handling.

When two costatements are requesting access to a single-user cofunction, the first request is honored and the second request is held. When loophead() notices that the first caller is not being called each time around the loop, it cancels the request, calls the abandonment code and allows the second caller in.

See Samples\Cofunc\Cofaband.c for sample code showing abandonment handling.

LIBRARY
COFUNC.LIB

void loopinit( void );

DESCRIPTION
This function should be called in the beginning of a program that uses single-user cofunctions. It initializes internal data structures that are used by loophead().

LIBRARY
COFUNC.LIB
lsqrt

unsigned int lsqrt( unsigned long x );

DESCRIPTION
Computes the square root of x. Note that the return value is an unsigned int. The fractional portion of the result is truncated.

PARAMETERS
x  long int input for square root computation

RETURN VALUE
Square root of x (fractional portion truncated).

LIBRARY
MATH.LIB

ltoa

char * ltoa( long num, char * ibuf )

DESCRIPTION
This function outputs a signed long number to the character array.

PARAMETERS
num Signed long number.
ibuf Pointer to character array.

RETURN VALUE
Pointer to the same array passed in to hold the result.

LIBRARY
STDIO.LIB

SEE ALSO
ltoa
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**

```c
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..`_fs.num_lx` 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 = 0xda;
    drive.part[partnum].endhead = 0xfe;
    drive.part[partnum].endseccyl = 0;
    drive.part[partnum].partsector = start;
    drive.part[partnum].partsecesize = ((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

Note: Starting with Dynamic C 9.01, this function BLOCKS!

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. (Valid prior to Dynamic C 9.01)

LIBRARY

PART.LIB
mbr_EnumDevice

mbr_EnumDevice( mbr_drvr *driver, mbr_dev *dev, int devnum, int (*checktype)() );

DESCRIPTION

This routine is called to learn about devices present on the driver passed in. The device will be added to the linked list of enumerated devices. Partition information will be filled in from the master boot record (MBR). Pointers to file system level partition information structures will be set to NULL.

PARAMETERS

driver Pointer to a DOS controller structure (setup during init of storage device driver.)
dev Pointer to a drive structure to be filled in.
devnum Physical device number of device on the driver.
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 device or structure.
-EINVAL if devnum invalid or does not exist.
-ENOMEM if memory for page buffer is not available.
-EUNFORMAT if the device is accessible, but not formatted. You can use it provided it is formatted/partitioned by either this library or another system.
-EBADPART if the partition table on the device is invalid
-ENOPART if the device does not have any sought partitions, If checktype parameter is NULL, this test is bypassed. This code is superseded by any other error detected.
-EXIST if the device has already been enumerated.
-EBUSY if the device is busy.

LIBRARY

PART.LIB
mbr_FormatDevice

int mbr_FormatDevice( mbr_dev * dev );

DESCRIPTION

Creates or rewrites the Master Boot Record on the device given. The routine will only rewrite
the Boot Loader code if an MBR already exists on the device. The existing partition table will
be preserved. To modify an existing partition table use mbr_CreatePartion.

Note: This routine is NOT PROTECTED from power loss and can make existing parti-
tions inaccessible if interrupted.

Note: This function is BLOCKING.

PARAMETERS

dev Pointer to MBR device structure

RETURN VALUE

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

LIBRARY

PART.LIB
mbr_MountPartition

int mbr_MountPartition( mbr_drive * drive, int pnum );

DESCRIPTION

Marks the partition as mounted. It is the higher level codes responsibility to verify that the
fs_part pointer for a partition is not in use (null) as this would indicate that another system
is in the process of mounting this device.

PARAMETERS

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
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
int mbr_ValidatePartitions( mbr_drive * drive );

DESCRIPTION

This routine will validate the partition table contained in the drive structure passed. It will verify
that all partitions fit within the bounds of the drive and that no partitions overlap.

PARAMETERS

drive Pointer to a drive structure

RETURN VALUE

0 for success
-EINVAL if the partition table in the drive structure is invalid.

LIBRARY

PART.LIB
**md5_append**

```c
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**

```c
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`
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 md5_state_t structure that was initialized by md5_init.
- **digest**: The 16-byte array that the hash value will be written into.

LIBRARY
MD5.LIB
**memchr**

**NEAR SYNTAX:** void * _n_memchr( void * src, int ch, unsigned int n );
**FAR SYNTAX:** void far * _f_memchr( void far * src, int ch, size_t n );

*Note:* By default, `memchr()` is defined to `_n_memchr()`.

**DESCRIPTION**

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

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

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

**PARAMETERS**

- **`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**

**NEAR SYNTAX:** int _n_memcmp( void *s1, void *s2, size_t n );

**FAR SYNTAX:** int _f_memcmp( void far *s1, void far *s2, size_t n );

**Note:** By default, `memcmp()` is defined to `_n_memcmp()`.

**DESCRIPTION**

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

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

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

**PARAMETERS**

- **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

NEAR SYNTAX: void * _n_memcpy( void *dst, void *src, unsigned int n );
FAR SYNTAX: void far * _f_memcpy( void far *dst, void far *src, size_t n );

Note: By default, memcpy() is defined to _n_memcpy().

DESCRIPTION
Copies a block of bytes from one destination to another. Overlap is handled correctly.
For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro USE_FAR_STRING will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with _f_strfunc where strfunc is the name of the string function.
Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the USE_FAR_STRING macro is defined and all pointers are near pointers, append _n_ to the function name, e.g., _n_strfunc. For more information about FAR pointers, see the Dynamic C User's Manual or the samples in Samples/Rabbit4000/FAR/.

PARAMETERS

<table>
<thead>
<tr>
<th>dst</th>
<th>Pointer to memory destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Pointer to memory source</td>
</tr>
<tr>
<td>n</td>
<td>Number of characters to copy</td>
</tr>
</tbody>
</table>

RETURN VALUE
Pointer to destination.

LIBRARY
STRING.LIB

SEE ALSO
memmove, memset
memmove

NEAR SYNTAX: void * _n_memmove( void *dst, void *src, unsigned int n );
FAR SYNTAX: _f_memmove( void far * dst, void far * src, size_t n);

Note: By default memmove() is defined to _n_memmove().

DESCRIPTION
Copies a block of bytes from one destination to another. Overlap is handled correctly.
For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the
function is called. The macro USE_FAR_STRING will change all calls to functions in this li-
brary to their far versions. The user may also explicitly call the far version with _f_strfunc
where strfunc is the name of the string function.

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

PARAMETERS

dst Pointer to 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**

**NEAR SYNTAX:** void * _n_memset( void * dst, int chr, unsigned int n );
**FAR SYNTAX:** void far * _f_memset( void far * dst, int chr, size_t n );

*Note:* By default, memset() is defined to _n_memset().

**DESCRIPTION**

Sets the first \( n \) bytes of a block of memory pointed to by \( \text{dst} \) to the character \( \text{chr} \).

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

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

**PARAMETERS**

- **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**

```c
unsigned long mktime( struct tm * timeptr );
```

**DESCRIPTION**

Converts the contents of structure pointed to by `timeptr` into seconds.

```c
struct tm {
    char tm_sec; // seconds 0-59
    char tm_min; // 0-59
    char tm_hour; // 0-23
    char tm_mday; // 1-31
    char tm_mon; // 1-12
    char tm_year; // 80-147 (1980-2047)
    char tm_wday; // 0-6 0==sunday
};
```

**PARAMETERS**

- `timeptr` Pointer to `tm` structure

**RETURN VALUE**

Time in seconds since January 1, 1980.

**LIBRARY**

`RTCLOCK.LIB`

**SEE ALSO**

- `mktime`, `tm_rd`, `tm_wr`
unsigned int mktm( struct tm * timeptr, unsigned long time );

DESCRIPTION
Converting 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
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
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 nf_device 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 nf_device structure for an initialized NAND flash device.

RETURN VALUE

The number of program pages on the NAND flash device.

LIBRARY

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

SEE ALSO

CalculateECC256, ChkCorrectECC256, xCalculateECC256, 
xChkCorrectECC256
long nf_getPageSize( nf_device * dev );

DESCRIPTION

Returns the size in bytes (excluding “spare” bytes) of each program page on the particular NAND flash device.

PARAMETERS

dev Pointer to an nf_device structure for an initialized NAND flash device.

RETURN VALUE

The number of data bytes in the NAND flash's program page, excluding the “spare” bytes used for ECC storage, etc.

LIBRARY

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

SEE ALSO

CalculateECC256, ChkCorrectECC256, xCalculateECC256, xChkCorrectECC256
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 runtime 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 dlm_initserialflash() in /LIB/RCM3300/downloadmanager.lib is where to look for code details. The upload utility is specifically for the RCM3300; however, even without the RCM3300, the utility is still useful in detailing what is necessary to manage multiple partitions.

The second mode of operation for NAND flash devices is direct access. An application that directly accesses the NAND flash (using calls such as nf_readPage() and nf_writePage()) may define NFLASH_USEERASEBLOCKSIZE to be either 0 (zero) or 1 (one) before NFLASH.LIB is #used, in order to set the NAND flash driver's main data program unit size to either the devices' program page size of 512 bytes or to its erase block size of 16 KB.

If not defined by the application, NFLASH_USEERASEBLOCKSIZE is set to the value 1 in NFLASH.LIB; this mode should maximize the NAND flash devices' life.

NFLASH_USEERASEBLOCKSIZE value 1 sets the driver up to program an erase block size at a time. This mode may be best for applications with only a few files open in write mode with larger blocks of data being written, and may be especially good at append operations. The trade off is reduced flash erasures at the expense of chunkier overhead due to the necessity of performing all 32 pages' ECC calculations for each programming unit written.

NFLASH_USEERASEBLOCKSIZE value 0 sets the driver up to program a program page size at a time. This mode may be best for applications with more than a few files open in write mode with smaller blocks of data being written, and may be especially good at interleaved file writes and/or random access write operations. The trade off is increased flash erasures with the benefit of spread out overhead due to the necessity of performing only 1 page's ECC calculations per programming unit written.
nf_initDevice (cont’d)

PARAMETERS

    dev  Pointer to an nf_device structure that will be filled in. An initialized
         nf_device struct acts as a handle for the NAND flash device.

    which  Number of the NAND flash device to initialize. Currently supported de-
           vice 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**

```c
int nf_InitDriver( mbr_drvr * driver, void * device_list );
```

**DESCRIPTION**

Initializes the NAND flash controller.

**PARAMETERS**

- **driver**
  Empty `mbr_drvr` 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 Rabbit Semiconductor website.

- **device_list**
  If not null, this is a pointer to the head of a linked list of `nf_device` structures for NAND flash devices that have each already been initialized by calling `nf_initDevice()`.
  If `device_list` is null, then this function attempts to initialize all NAND flash devices and provide a default linked list of `nf_device` structures in order from device number 0 on up. If the initialization of a NAND flash device is unsuccessful, then its `nf_device` structure is not entered into the linked list.

**RETURN VALUE**

- **0**: Success
- **<0**: Negative value of a FAT file system error code

**LIBRARY**

`NFLASH_FAT.LIB` (This function was introduced in Dynamic C 9.01)
nf_isBusyRBHW

int nf_isBusyRBHW( nf_device * dev );

DESCRIPTION

Returns 1 if the specified NAND flash device is busy. Uses the hardware Ready/Busy check method, and can be used to determine the device's busy status even at the start of a read page command. Note that this function briefly enforces the Ready/Busy input port bit, reads the pin status, and then restores the port bit to its previous input/output state. There should be little or no visible disturbance of the LED output which shares the NAND flash's Ready/Busy status line.

PARAMETERS

dev Pointer to an initialized nf_device 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
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 nf_device 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 nf_device structure
- **buffer**: 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
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 nf_device structure
- **buffer**: 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
nf_XD_Detect

long nf_XD_Detect( int debounceMode );

DESCRIPTION

This function attempts to read the xD card ID and searches the internal device table for that ID in detect mode 1. In detect mode 0 it just uses the xD card detect.

Assumes only one XD card present.

WARNING! - This should not be called to determine if it is safe to do write operations if there is a chance a removable device might be pulled between calling it and the write. It is best used to determine if a device is present to proceed with an automount after a device has been un-mounted in SW and removed.

PARAMETERS

debounceMode

0 - no debouncing
1 - busy wait for debouncing interval
2 - for use if function to be called until debouncing interval is done, e.g.,

\[
\text{waitfor} (rc = \text{nfx}_\text{XD}_\text{Detect}(1) != \text{EAGAIN});
\]

-EAGAIN will be returned until done.

RETURN VALUE

>0: The ID that was found on the device and in the table

-EBUSY: NAND flash device is busy

-ENODEV: No device found

-EAGAIN: if debounceMode equals 2, then not done debouncing, try again

LIBRARY

NFLASH_FAT.LIB
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
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( void );

DESCRIPTION
Enter a critical section. Interrupts will be disabled until OS_EXIT_CRITICAL() is called. Task switching is disabled. This function must be used with great care, since misuse can greatly increase the latency of your application. Note that nesting OS_ENTER_CRITICAL() calls will work correctly.

LIBRARY
UCOS2.LIB

OS_EXIT_CRITICAL

void OS_EXIT_CRITICAL( void );

DESCRIPTION
Exit a critical section. If the corresponding previous OS_ENTER_CRITICAL() call disabled interrupts (that is, interrupts were not already disabled), then interrupts will be enabled. Otherwise, interrupts will remain disabled. Hence, nesting calls to OS_ENTER_CRITICAL() will work correctly.

LIBRARY
UCOS2.LIB
OSFlagAccept

OS_FLAGS OSFlagAccept( OS_FLAG_GRP * pgrp, OS_FLAGS flags, INT8U wait_type, INT8U * err );

DESCRIPTION

This function is called to check the status of a combination of bits to be set or cleared in an event flag group. Your application can check for ANY bit to be set/cleared or ALL bits to be set/cleared.

This call does not block if the desired flags are not present.

PARAMETERS

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 flags should be 0x03.

wait_type 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:

• OS_FLAG_WAIT_CLR_ALL - You will check ALL bits in flags to be clear (0)
• OS_FLAG_WAIT_CLR_ANY - You will check ANY bit in flags to be clear (0)
• OS_FLAG_WAIT_SET_ALL - You will check ALL bits in flags to be set (1)
• OS_FLAG_WAIT_SET_ANY - You will check ANY bit in flags 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 (cont’d)**

**err**

Pointer to an error code. Possible values are:

- **OS_NO_ERR** - No error
- **OS_ERR_EVENT_TYPE** - Not pointing to an event flag group
- **OS_FLAG_ERR_WAIT_TYPE** - Proper wait_type argument not specified.
- **OS_FLAG_INVALID_PGRP** - null pointer passed instead of the event flag group handle.
- **OS_FLAG_ERR_NOT_RDY** - Flags not available.

**RETURN VALUE**

The state of the flags in the event flag group.

**LIBRARY**

OS_FLAG.C (Prior to DC 8:UCOS2.LIB)
OSFlagCreate

OS_FLAG_GRP * OSFlagCreate( OS_FLAGS flags, INT8U * err );

DESCRIPTION
This function is called to create an event flag group.

PARAMETERS
- **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:
  - 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:UCOSZ.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:
- 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:
- 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 flags should be 0x03.
wait_type Specifies whether you want ALL bits to be set or ANY of the bits to be set. You can specify the following argument:

• OS_FLAG_WAIT_CLR_ALL - You will wait for ALL bits in mask to be clear (0)
• OS_FLAG_WAIT_SET_ALL - You will wait for ALL bits in mask to be set (1)
• OS_FLAG_WAIT_CLR_ANY - You will wait for ANY bit in mask to be clear (0)
• OS_FLAG_WAIT_SET_ANY - You will wait for ANY bit in mask to be set (1)

Note: Add OS_FLAG_CONSUME 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 wait_type parameter to:

OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME

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 (cont’d)**

**err**

Pointer to an error code. Possible values are:

- **OS_NO_ERR** - The desired bits have been set within the specified time-out.
- **OS_ERR_PEND_ISR** - If you tried to PEND from an ISR.
- **OS_FLAG_INVALID_PGRP** - If pgrp is a null pointer.
- **OS_ERR_EVENT_TYPE** - You are not pointing to an event flag group
- **OS_TIMEOUT** - The bit(s) have not been set in the specified time-out.
- **OS_FLAG_ERR_WAIT_TYPE** - You didn't specify a proper wait_type argument.

**RETURN VALUE**

The new state of the flags in the event flag group when the task is resumed or, 0 if a timeout or an error occurred.

**LIBRARY**

- **OS_FLAG.C** (Prior to DC 8:UCOS2.LIB)
OSFlags 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

If opt (see below) is OS_FLAG_SET, 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:

0x31 (note, bit 0 is least significant bit)

If opt (see below) is OS_FLAG_CLR, 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:

0x31 (note, bit 0 is least significant bit)

opt

Indicates whether the flags will be:

set (OS_FLAG_SET), or cleared (OS_FLAG_CLR)

err

Pointer to an error code. Valid values are:

• 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
• OS_FLAG_INVALID_OPT - 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:
            • 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

```c
void OSInit( void );
```

**DESCRIPTION**

Initializes µC/OS-II data; must be called before any other µC/OS-II functions are called.

**LIBRARY**

`UCOS2.LIB`

**SEE ALSO**

`OSTaskCreate, OSTaskCreateExt, OSStart`

OSMboxAccept

```c
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:
• OS_DEL_NO_PEND - Delete mailbox only if no task pending
• OS_DEL_ALWAYS - 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:
• OS_NO_ERR - Call was successful; mailbox was deleted
• OS_ERR_DEL_ISR - Attempt to delete mailbox from ISR
• OS_ERR_INVALID_OPT - Invalid option was specified
• OS_ERR_TASK_WAITING - One or more tasks were waiting on the mailbox
• OS_ERR_EVENT_TYPE - No pointer passed to a mailbox
• OS_ERR_PEVENT_NULL - If pevent is a null pointer.

RETURN VALUE

!= (void *)0 Is a pointer to the event control clock (OS EVENT) associated with the created mailbox

== (void *)0 If no event control blocks were available

LIBRARY

OS_MBOX.C
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**: Pointer to a variable for holding an error code. Possible error messages are:
  - **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
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:
  
  - **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)
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 μC/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

```c
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
INT8U OSMutexAccept( OS_EVENT * pevent, INT8U * err );

DESCRIPTION
This function checks the mutual exclusion semaphore to see if a resource is available. Unlike OSMutexPend(), OSMutexAccept() does not suspend the calling task if the resource is not available or the event did not occur. This function cannot be called from an ISR because mutual exclusion semaphores are intended to be used by tasks only.

PARAMETERS
pevent  Pointer to the event control block.
err  Pointer to an error code that will be returned to your application:
  • OS_NO_ERR - if the call was successful.
  • OS_ERR_EVENT_TYPE - if pevent is not a pointer to a mutex
  • OS_ERR_PEVENT_NULL - pevent is a null pointer
  • OS_ERR_PEND_ISR - if you called this function from an ISR

RETURN VALUE
1: Success, the resource is available and the mutual exclusion semaphore is acquired.

0: Error, either the resource is not available, or you didn't pass a pointer to a mutual exclusion semaphore, or you called this function from an ISR.

LIBRARY
OS_MUTEX.C
OSMutexCreate

OS_EVENT *OSMutexCreate( INT8U prio, INT8U * err );

DESCRIPTION

This function creates a mutual exclusion semaphore. Note that:

• The LEAST significant 8 bits of the OSEventCnt field of the mutex’s event control block are
  used to hold the priority number of the task owning the mutex or 0xFF if no task owns the
  mutex.

• The MOST significant 8 bits of the OSEventCnt field of the mutex’s event control block are
  used to hold the priority number to use to reduce priority inversion.

PARAMETERS

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 pri-
       ority that is LOWER in value than ANY of the tasks competing for the mu-
       tex.

err  Pointer to error code that will be returned to your application:

  • 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 ex-
    ist.
  • OS_ERR_PEVENT_NULL - no more event control blocks available.
  • OS_PRIO_INVALID - if the priority you specify is higher that the max-
    imum 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:

• OS_DEL_NO_PEND - Delete mutex only if no task pending
• OS_DEL_ALWAYS - 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:

• OS_NO_ERR - The call was successful and the mutex was deleted
• OS_ERR_DEL_ISR - Attempted to delete the mutex from an ISR
• OS_ERR_INVALID_OPT - An invalid option was specified
• OS_ERR_TASK_WAITING - One or more tasks were waiting on the mutex
• OS_ERR_EVENT_TYPE - If you didn't pass a pointer to a mutex point-
er.

RETURN VALUE

pevent On error.

(OS_EVENT *) 0 Mutex was deleted.

LIBRARY

OS_MUTEX.C
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 Pointer to where an error message will be deposited. Possible error messages are:

- OS_NO_ERR - The call was successful and your task owns the mutex
- OS_TIMEOUT - The mutex was not available within the specified time.
- OS_ERR_EVENT_TYPE - If you didn't pass a pointer to a mutex
- OS_ERR_PEVENT_NULL - pevent is a null pointer
- OS_ERR_PEND_ISR - If you called this function from an ISR and the result would lead to a suspension.

LIBRARY
OS_MUTEX.C
OSMutexPost

`INT8U OSMutexPost( OS_EVENT * pevent );`

DESCRIPTION
This function signals a mutual exclusion semaphore.

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pevent</td>
<td>Pointer to mutex’s event control block.</td>
</tr>
</tbody>
</table>

RETURN VALUE

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS_NO_ERR</td>
<td>The call was successful and the mutex was signaled.</td>
</tr>
<tr>
<td>OS_ERR_EVENT_TYPE</td>
<td>If you didn't pass a pointer to a mutex</td>
</tr>
<tr>
<td>OS_ERR_PEVENT_NULL</td>
<td><code>pevent</code> is a null pointer</td>
</tr>
<tr>
<td>OS_ERR_POST_ISR</td>
<td>Attempted to post from an ISR (invalid for mutexes)</td>
</tr>
<tr>
<td>OS_ERR_NOT_MUTEX_OWNER</td>
<td>The task that did the post is NOT the owner of the MUTEX.</td>
</tr>
</tbody>
</table>

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
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 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:
• OS_DEL_NO_PEND - Delete queue only if no task pending
• OS_DEL_ALWAYS - 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:
• OS_NO_ERR - Call was successful and queue was deleted
• OS_ERR_DEL_ISR - Attempt to delete queue from an ISR
• OS_ERR_INVALID_OPT - Invalid option was specified
• OS_ERR_TASK_WAITING - One or more tasks were waiting on the queue
• OS_ERR_EVENT_TYPE - You didn't pass a pointer to a queue
• OS_ERR_PEVENT_NULL - If pevent is a null pointer.

RETURN VALUE

pevent Error

(OS_EVENT *) 0 The queue was successfully deleted.

LIBRARY

OS_Q.C (Prior to DC 8:UCOS2.LIB)
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
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, OSQuery
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
INT8U OSQPostFront( OS_EVENT * pevent, void * msg );

DESCRIPTION

Sends a message to the specified queue, but unlike OSQPost(), the message is posted at the front instead of the end of the queue. Using OSQPostFront() allows 'priority' messages to be sent.

PARAMETERS

pevent Pointer to message queue’s event control block.
msg Pointer to the message to send. A null pointer must not be sent.

RETURN VALUE

OS_NO_ERR The call was successful and the message was sent.
OS_Q_FULL The queue cannot accept any more messages because it is full.
OS_ERR_EVENT_TYPE A pointer to a queue was not passed.
OS_ERR_PEVENT_NULL If pevent is a null pointer.
OS_ERR_POST_NULL_PTR Attempting to post to a non mailbox.

LIBRARY

OS_Q.C (Prior to DC 8:UCOS2.LIB)

SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPend, OSQPost, OSQuery
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 capabil-
ity 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 inter-
rupt 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:
• OS_POST_OPT_NONE - POST to a single waiting task (Identical to
  OSQPost())
• OS_POST_OPT_BROADCAST - POST to ALL tasks that are waiting on
  the queue
• OS_POST_OPT_FRONT - POST as LIFO (Simulates
  OSQPostFront())

The last 2 flags may be combined:
• OS_POST_OPT_FRONT + OS_POST_OPT_BROADCAST - is identi-
cal to OSQPostFront() except that it will broadcast msg to all wait-
ing tasks.

RETURN VALUE

OS_NO_ERR The call was successful and the message was sent.

OS_Q_FULL The queue is full, cannot accept any more messages.

OS_ERR_EVENT_TYPE A pointer to a queue was not passed.

OS_ERR_PEVENT_NULL If pevent is a null pointer.

OS_ERR_POST_NULL_PTR Attempting to post a null pointer.

LIBRARY

OS_Q.C (Prior to DC 8:UCOS2.LIB)
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

```c
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

```c
void OSSchedUnlock( void );
```

**DESCRIPTION**

Allow task rescheduling. There must be a matched call to `OSSchedUnlock()` for every call to `OSSchedLock()`.

**LIBRARY**

`UCOS2.LIB`

**SEE ALSO**

`OSSchedLock`
OSSemAccept

INT16U OSSemAccept( OS_EVENT * pevent );

DESCRIPTION
This function checks the semaphore to see if a resource is available or if an event occurred. Unlike OSSemPend(), OSSemAccept() does not suspend the calling task if the resource is not available or the event did not occur.

PARAMETERS
pevent Pointer to the desired semaphore’s event control block

RETURN VALUE
Semaphore value:
If >0, semaphore value is decremented; value is returned before the decrement.
If 0, then either resource is unavailable, event did not occur, or null or invalid pointer was passed to the function.

LIBRARY
UCOS2.LIB

SEE ALSO
OSSemCreate, OSSemPend, OSSemPost, OSSemQuery
OSSemCreate

OS_EVENT * 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 OSSemPend( OS_EVENT * pevent, INT16U timeout, INT8U * err );

DESCRIPTION
Waits on a semaphore.

PARAMETERS
  pevent  Pointer to the desired semaphore’s event control block
  timeout  Time in clock ticks to wait for the resource. If 0, the task will wait until the resource becomes available or the event occurs.
  err  Pointer to error message.

LIBRARY
UCOS2.LIB

SEE ALSO
OSSemAccept, OSSemCreate, OSSemPost, OSSemQuery
INT8U OSSemPost( OS_EVENT * pevent );

DESCRIPTION

This function signals a semaphore.

PARAMETERS

pevent Pointer to the desired semaphore’s event control block

RETURN VALUE

OS_NO_ERR The call was successful and the semaphore was signaled.
OS_SEM_OVF If the semaphore count exceeded its limit. In other words, you have signalled the semaphore more often than you waited on it with either OSSemAccept() or OSSemPend().
OS_ERR_EVENT_TYPE If a pointer to a semaphore not passed.
OS_ERR_PEVENT_NULL If pevent is a null pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemAccept, OSSemCreate, OSSemPend, OSSemQuery
INT8U OSSemQuery( 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 pevent parameter is a null pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemAccept, OSSemCreate, OSSemPend, OSSemPost
**OSSetTickPerSec**

```c
INT16U OSSetTickPerSec( 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 that 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**

```c
void OSStart( void );
```

**DESCRIPTION**

Starts the multitasking process, allowing µC/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**

```c
void OSStatInit( void );
```

**DESCRIPTION**
Determines CPU usage.

**LIBRARY**
UCOS2.LIB

---

**OSTaskChangePrio**

```c
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 µC/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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>Pointer to the task’s starting address.</td>
</tr>
<tr>
<td>pdata</td>
<td>Pointer to a task’s initial parameters.</td>
</tr>
<tr>
<td>stk_size</td>
<td>Number of bytes of the stack.</td>
</tr>
<tr>
<td>prior</td>
<td>The task’s unique priority number.</td>
</tr>
</tbody>
</table>

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 µC/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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>Pointer to task’s code.</td>
</tr>
<tr>
<td>pdata</td>
<td>Pointer to optional data area; used to pass parameters to the task at start of execution.</td>
</tr>
<tr>
<td>prio</td>
<td>The task’s unique priority number; the lower the number the higher the priority.</td>
</tr>
<tr>
<td>id</td>
<td>The task’s identification number (0...65535).</td>
</tr>
<tr>
<td>stk_size</td>
<td>Size of the stack in number of elements. If OS_STK is set to INT8U, stk_size corresponds to the number of bytes available. If OS_STK is set to INT16U, stk_size contains the number of 16-bit entries available. Finally, if OS_STK is set to INT32U, stk_size contains the number of 32-bit entries available on the stack.</td>
</tr>
<tr>
<td>pext</td>
<td>Pointer to a user-supplied Task Control Block (TCB) extension.</td>
</tr>
<tr>
<td>opt</td>
<td>The lower 8 bits are reserved by µC/OS-II. The upper 8 bits control application-specific options. Select an option by setting the corresponding bit(s).</td>
</tr>
</tbody>
</table>

RETURN VALUE

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS_NO_ERR</td>
<td>The call was successful.</td>
</tr>
<tr>
<td>OS_PRIO_EXIT</td>
<td>Task priority already exists (each task MUST have a unique priority).</td>
</tr>
<tr>
<td>OS_PRIO_INVALID</td>
<td>The priority specified is higher than the maximum allowed (i.e. ≥OS_LOWEST_PRIO).</td>
</tr>
</tbody>
</table>

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreate
void OSTaskCreateHook( OS_TCB * ptcb );

DESCRIPTION
Called by µC/OS-II whenever a task is created. This call-back function resides in
UCOS2.LIB and extends functionality during task creation by allowing additional informa-
tion 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, there-
fore, 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 µC/OS-II’s idle task.
OS_PRIO_INVALID The priority specified is higher than the maximum allowed (i.e. ≥ OS_LOWEST_PRIO) or, OS_PRIO_SELF 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
void OSTaskDelHook( OS_TCB * ptcb );

DESCRIPTION
Called by μC/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
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

```c
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

```c
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**

```c
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**

```c
void OSTaskStatHook( void );
```

**DESCRIPTION**

Called every second by µC/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`
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
INT8U OSTaskSuspend( INT8U prio );

DESCRIPTION
Suspend 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. ≥ OS_LOWEST_PRIO) or, OS_PRIO_SELF is not specified.
OS_TASK_SUS_PRIO The task to suspend does not exist.

LIBRARY
UCOS2.LIB

SEE ALSO
OSTaskResume
### OSTaskSwHook

```c
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

```c
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

```c
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`
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
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 oc-
curred (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. ≥
OS_LOWEST_PRIO).
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**

```c
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**

```c
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**

```c
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**

```c
INT16U OSVersion( void );
```

**DESCRIPTION**

Returns the version number of µC/OS-II. The returned value corresponds to µC/OS-II's version number multiplied by 100; i.e., version 2.00 would be returned as 200.

**RETURN VALUE**

Version number multiplied by 100.

**LIBRARY**

UCOS2.LIB

---

**outchrs**

```c
char outchrs( 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 `putc` should take a character argument.

**RETURN VALUE**

The character in parameter `c`.

**LIBRARY**

STDIO.LIB

**SEE ALSO**

outstr
**outstr**

```c
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**

- **string**: String to output
- **putc**: Routine to output one character. The function pointed to by `putc` should take a character argument.

**RETURN VALUE**

Pointer to null at end of string.

**LIBRARY**

`STDC.IO.LIB`

**SEE ALSO**

`outchrs`
paddr

unsigned long paddr( void *pointer );

DESCRIPTION
Determines the physical address of a logical pointer. This function is compatible with both shared and separate I&D space compile modes. Use caution when converting a pointer in the xmem window, i.e., in the range 0xE000 to 0xFFFF, as this function will return the physical address based on the XPC on entry.

PARAMETERS

pointer The pointer to convert.

RETURN VALUE
The physical address of the entity that the pointer is referencing.

LIBRARY
XMEM.LIB

SEE ALSO
paddrDS, paddrSS
**paddrDS**

```c
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
The paddrSS function is deprecated and has been replaced by a macro redefining it to simply paddr. Refer to the paddr function help for usage information.

PARAMETERS
pointer The pointer to convert.

RETURN VALUE
The physical address of the entity that the pointer is referencing.

LIBRARY
XMEM.LIB

SEE ALSO
paddr, paddrDS
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**

```c
xmeme 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**

```c
ld ix,my_pool
lcall palloc_fast
jr c,.no_free
; HL points to element
```

**PARAMETERS**

- `p` Pool handle structure, as previously passed to `pool_init()`. Pass this in IX.

**RETURN VALUE**

- C flag set: no free elements were available.
- C flag clear (NC): HL points to an element.

If the pool is not linked, your application can use this element provided it does not write more than `p->elsize` bytes to it (this was the `elsize` parameter passed to `pool_init()`). If the pool is linked, you can write `p->elsize-4` bytes to it.

**LIBRARY**

POOL.LIB

**SEE ALSO**

`pool_init`, `pfree_fast`, `pavail_fast`, `palloc`
word pavail(Pool_t * p);

DESCRIPTION
Return the number of elements that are currently available for allocation.

PARAMETERS
p         Pool handle structure, as previously passed to pool_init() or pool_xinit().

RETURN VALUE
Number of elements available for allocation.

LIBRARY
POOL.LIB

SEE ALSO
pool_init, pool_xinit, phwm, pnel
pavail_fast

xmem word pavail_fast( Pool_t * p );

DESCRIPTION

Return the number of elements that are currently available for allocation.

This is an assembler-only version of pavail().

*** Do _not_ call this function from C. ***

REGISTERS

Parameter in IX
Trashes F, DE
Return value in HL, Z flag

EXAMPLE

ld ix,my_pool
lcall pavail_fast
; HL contains number of available elements

PARAMETERS

p Pool handle structure, as previously passed to pool_init() or pool_xinit(). This must be provided in the IX register.

RETURN VALUE

Number of elements available for allocation. The return value is placed in HL. In addition, the 'Z' flag is set if there are no free elements.

LIBRARY

POOL.LIB

SEE ALSO

pool_init, pool_xinit, phwm, pnel
pcalloc

```c
void * pcalloc( 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`
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:

\[ \text{pool_link}(p, \text{<non-zero>}) \]

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
xmem void * pfirst_fast( Pool_t * p );

DESCRIPTION
Get the first allocated element in a root pool. The pool MUST be set to being a linked pool by using:

    pool_link(p, <non-zero>);

Otherwise the results are undefined.
This is an assembler-only version of pfirst().
*** Do _not_ call this function from C. ***

REGISTERS
Parameter in IX
Trashes F, DE
Return value in HL, carry flag

EXAMPLE

    ld ix,my_pool
    lcall pfirst_fast
    jr c,.no elems
    ; HL points to first element

PARAMETERS
p Pool handle structure, as previously passed to pool_init(). Pass this in the IX register.

RETURN VALUE
C flag set, HL=0: There are no allocated elements.
C flag clear (NC): HL points to first element.

LIBRARY
POOL.LIB

SEE ALSO
pool_init, pool_link, pfirst, pnext_fast
pfree

void pfree( Pool_t * p, void * e );

DESCRIPTION
Free an element that was obtained via palloc(). Note: if you free an element that was not
allocated from this pool, or was already free, or was outside the pool, then your application will
-crash! You can detect most of these programming errors by defining the following symbols be-
fore #use pool.lib:

POOL_DEBUG
POOL_VERBOSE

PARAMETERS
p Pool handle structure, as previously passed to palloc().
e Element to free, which was returned from palloc().

RETURN VALUE
None

LIBRARY
POOL.LIB

SEE ALSO
pool_init, palloc, pcaploc, phwm, pavail
pmem void pfree_fast( Pool_t * p, void * e );

DESCRIPTION
Free an element that was previously obtained via \texttt{palloc()}. This is an assembler-only version of \texttt{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
\begin{verbatim}
ld ix,my_pool
ld de,(element_addr)
lcall pfree_fast
\end{verbatim}

PARAMETERS
\begin{itemize}
\item \texttt{p} Pool handle structure, as previously passed to \texttt{pool_alloc()} or \texttt{palloc_fast}. This must be in the IX register.
\item \texttt{e} Element to free, which was returned from \texttt{palloc()}. This must be in the DE register.
\end{itemize}

RETURN VALUE
None

LIBRARY
POOL.LIB

SEE ALSO
pool_init, palloc_fast, pavail_fast, pxfree_fast
word phwm( Pool_t * p );

DESCRIPTION

Return the largest number of elements ever simultaneously allocated from the given pool, i.e., the pool high water mark.

You can use this function to help size a pool, since it may be difficult to determine the optimum number of elements without running a trial program.

PARAMETERS

p        Pool handle structure, as previously passed to pool_init() or pool_xinit().

RETURN VALUE

Maximum number of elements ever allocated.

LIBRARY

POOL.LIB

SEE ALSO

pool_init, pool_xinit, pavail
**pktXclose**

```c
void pktXclose( void ); /* X is A-F */
```

**DESCRIPTION**

Disables serial port X. The functions `pktEclose()` and `pktFclose()` may be used with the Rabbit 3000 and Rabbit 4000.

**LIBRARY**

`PACKET.LIB`

**pktXgetErrors**

```c
char pktXgetErrors( void ); /* X is A-F */
```

**DESCRIPTION**

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

The functions `pktEgetErrors()` and `pktFgetErrors()` may be used with the Rabbit 3000 and Rabbit 4000.

**RETURN VALUE**

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

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

**LIBRARY**

`PACKET.LIB`
int pktXinitBuffers( int buf_count, int buf_size ); /* X is A-F */

DESCRIPTION

Allocates extended memory for channel X receive buffers. This function should not be called more than once in a program. The total memory allocated is buf_count*(buf_size + 2) bytes.

The functions pktEinitBuffers() and pktFinitBuffers() may be used with the Rabbit 3000 and Rabbit 4000.

PARAMETERS

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
int pktXopen( long baud, int mode, char options, int (*test_packet)() ); /* X is A-F */

DESCRIPTION

Opens serial port X. The functions pktEopen() and pktFopen() may be used with the Rabbit 3000 and Rabbit 4000.

The packet driver is meant to be used with a variety of transceiver hardware, so some functions must be defined by the user. Each of these functions, listed below, take no arguments and return nothing.

• pktXinit() - Initializes the communication hardware. Called inside pktXopen(). This function may be written in C. It will only be called once each time the packet driver is opened, so speed is not a major concern. This is where I/O pins should be configured and any other setup should be performed.

• pktXrx() - Sets the hardware to receive data. This function must be written in assembly. Any registers besides the 8-bit accumulator A must be preserved first, and restored before returning. This function is called when the driver switches from transmit to receive mode once there are no packets to send. This function is necessary for half-duplex connections and other types of shared bus schemes so that the transmitter can be disabled, allowing other nodes to use the lines.

• pktXtx() - Sets the hardware to transmit data. This function must be written in assembly. The same rules for register usage as for pktXrx() apply. This function is called whenever the driver switches from receive to transmit mode in response to an additional packet or packets being available for sending. A typical use of this function is to enable any necessary transmitter hardware.

See the sample program Samples/PKTDEMO.C for an example of how to write these user-supplied functions. See technical note TN213 “Rabbit Serial Port Software” for more information on the packet driver.
pktXopen (cont’d)

PARAMETERS

baud Bits per second of data transfer: minimum is 2400.

mode Type of packet scheme used, the options are:
  • PKT_GAPMODE
  • PKT_9BITMODE
  • PKT_CHARMODE

options Further specification for the packet scheme. The value of this depends on
  the mode used:
  • gap mode - minimum gap size (in byte times)
  • 9-bit mode - type of 9-bit protocol
    • PKT_RABBITSTARTBYTE
    • PKT_LOWSTARTBYTE
    • PKT_HIGHSTARTBYTE
  • char mode - character marking start of packet

test_packet Pointer to a function that tests for completeness of a packet. The function
  should return 1 if the packet is complete, or 0 if more data should be read
  in. For gap mode the test function is not used and should be set to null.

RETURN VALUE

  1: The Rabbit’s bps setting is within 5% of the input baud.
  0: The Rabbit’s bps setting differs by more than 5% of the input baud

LIBRARY

  PACKET.LIB
int pktXreceive( void * buffer, int buffer_size ); /* X is A-F */

DESCRIPTION

Gets a received packet, if there is one, from serial port X.

The functions pktEreceive() and pktFreceive() may be used with the Rabbit 3000 and Rabbit 4000.

PARAMETERS

buffer A buffer for the packet to be written into.

buffer_size Length of the data buffer.

RETURN VALUE

>0: Number of bytes in the successfully received packet.

0: No new packet has been received.

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

-2: A needed test_packet function is not defined.

LIBRARY

PACKET.LIB
**pktXsend**

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

**DESCRIPTION**

Initiates the sending of a packet of data using serial port X. This function will always return immediately. If there is already a packet being transmitted, this call will return 0 and the packet will not be transmitted, otherwise it will return 1.

`pktXsending()` checks if the packet is done transmitting. The system will be using the buffer until then.

The functions `pktEsend()` and `pktFsend()` may be used with the Rabbit 3000 and Rabbit 4000.

**PARAMETERS**

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

DESCRIPTION
Tests if a packet is currently being sent on serial port X. If pktXsending() returns true, the transmitter is busy and cannot accept another packet.

The functions pktEsetParity() and pktFsetParity() may be used with the Rabbit 3000 and Rabbit 4000.

RETURN VALUE
1: A packet is being transmitted.
0: Port X is idle, ready for a new packet.

LIBRARY
PACKET.LIB

void pktXsetParity( char mode ); /* X is A-F */

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

The functions pktEsetParity() and pktFsetParity() may be used with the Rabbit 3000 and Rabbit 4000.

PARAMETERS

mode
Code for mode of parity bit:
- 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
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

```c
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**

```assembly
ld ix,my_pool
lcall plast_fast
jr c,.no elems
; HL points to last element
```

**PARAMETERS**

`p`  
Pool handle structure, as previously passed to pool_init() . Pass this in IX register.

**RETURN VALUE**

C flag set, HL=0: there are no allocated elements
C flag clear (NC): HL points to last element.

**LIBRARY**

POOL.LIB

**SEE ALSO**

pool_init, pool_link, plast, pprev_fast
void * pmovbetween( 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:

\[ \text{pprev}(p, f) == d \]

and

\[ \text{pnext}(p, d) == f \]

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:

\[ \text{pool_link}(p, <\text{non-zero}>) \]

Otherwise the results are undefined.

PARAMETERS

- **p**
  Pool handle structure, as previously passed to pool_init().

- **e**
  Address of element to move, obtained by, e.g., plast(). 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 plast() 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 \( e == d \) or \( e == f \), 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 \( \text{pnext}(p, d) == f \). Otherwise, null is returned. If this parameter is null, then “f” must point to the first element in the list, and “e” is inserted at the start of the list.
pmovebetween (cont’d)

Second reference element. The element “e” will be inserted before this element. On entry, it must be true that \( \text{pprev}(p, f) == d \). 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:

```c
    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 \( \text{plast}(p) \), 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**

```c
    void * d, * e, * f;

    e = plast(p);    // element to move
    f = pnext(p, d = pfirst(p));    // d, f are first 2 elements
    pmovebetween(p, e, d, f);
```

**LIBRARY**

POOL.LIB

**SEE ALSO**

`pool_init`, `pool_link`, `plast`, `pfirst`, `pnext`, `pprev`, `preorder`
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 pool_init(). Pass in IX register
- e Address of element to move. Pass in DE 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
word pnel( Pool_t * p );

DESCRIPTION

Return the number of elements that are in the pool, both free and used. This includes elements appended using pool_append() etc.

PARAMETERS

p

Pool handle structure, as previously passed to pool_init() or pool_xinit().

RETURN VALUE

Number of elements total

LIBRARY

POOL.LIB

SEE ALSO

pool_init, pool_xinit, pavail
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:

```c
void * e;
Pool_t * p;
for (e = pfirst(p); e; e = pnext(p, e)) {
  ...
}
```

PARAMETERS

<table>
<thead>
<tr>
<th>p</th>
<th>Pool handle structure, as previously passed to pool_init().</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Previous element address, obtained by, e.g., pfirst(). This must be an allocated element in the given pool; otherwise, the results are undefined. Be careful when iterating through a list and deleting elements using pfree(): once the element is deleted, it is no longer valid to pass its address to this function. If this parameter is null, then the result is the same as pfirst(). This ensures the invariant pnext(p, pprev(p, e)) == e.</td>
</tr>
</tbody>
</table>

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

```c
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**

```asm
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 `pool_init()`. Pass this in IX register.

- **e**
  
  Current element, address in DE register. See `pnext()` 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`
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 \(x\).

RETURN VALUE
The polynomial value.

LIBRARY
MATH.LIB
### pool_append

```c
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 `pool_init()`.

- **base**
  
  Base address of the root data memory area to append to this pool. This must be `nel*elsize` bytes long. Typically, this would be a static (global) array.

- **nel**
  
  Number of elements in the memory area. The sum of `nel` 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`
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 Pool_t.

base  Base address of the root data memory area to be managed in this pool. This must be nel*elsize 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
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 pool_init() or pool_xinit().

link  Must be one of the following:

• POOL_NOT_LINKED (0): the pool is not to be linked.
• POOL_LINKED_AUTO (1): the pool is linked, and newly allocated elements are always added at the end of the list.
• POOL_LINKED_BY_APP (2): the pool is linked, but newly allocated elements are not added to the list. The application must call preorder() or pmovebetween() 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
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 pool_xinit().

base

Base address of the xmem data memory area to append to this pool. This must be nel*elsize bytes long. Typically, this would be an area allocated using xalloc().

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**

```c
float pow( float x, float y );
```

**DESCRIPTION**

Raises $x$ to the $y$th power.

**PARAMETERS**

- $x$ Value to be raised
- $y$ Exponent

**RETURN VALUE**

$x$ to the $y$th power

**LIBRARY**

MATH.LIB

**SEE ALSO**

exp, pow10, sqrt

---

**pow10**

```c
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
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 \text{fftreal}(), the imaginary part of the \( X[0] \) term (stored \( x[1] \)) will contain the real part of the \( f_{\text{max}} \) term and will affect the calculation of the dc power. If the dc power or the \( f_{\text{max}} \) power is important, the \( f_{\text{max}} \) term should be retrieved from \( x[1] \) and \( x[1] \) set to zero before calling \text{powerspectrum}().

The power of the \( k \)th term can be retrieved via

\[ P[k]=*{(long*) &x[2k]}*2^\text{blockexp}. \]

The value of \text{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 \( x \).

\( \text{blockexp} \) Pointer to integer block exponent.

LIBRARY

\text{FFT.LIB}

SEE ALSO

\text{fftcplx, fftcplxinv, fftreal, fftrealinv, hanncplx, hannreal}
void * pprev( Pool_t * p, void * e );

DESCRIPTION
Get the previously allocated element in a root pool. The pool MUST be set to being a linked pool using pool_link(p, <non-zero>); otherwise, the results are undefined.

You can easily iterate through all of the allocated elements of a root pool using the following construct:

void * e;
Pool_t * p;

for (e = plast(p); e; e = pprev(p, e)) {
    ...
}

PARAMETERS
p Pool handle structure, as previously passed to pool_init().

e Previous element address, obtained by, e.g., plast(). This must be an allocated element in the given pool; otherwise, the results are undefined. Be careful when iterating through a list and deleting elements using pfree(): once the element is deleted, it is no longer valid to pass its address to this function. If this parameter is null, then the result is the same as plast(). This ensures the invariant

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
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 pool_init(). Pass this in IX register.

e Current element, address in DE register. See pprev() 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
void * pputlast(Pool_t * p, void * e);

DESCRIPTION
Atomically remove allocated element “e” and re-insert it at the end of the allocated list. “Atomically” means that the POOL_IPSET level is used to lock out other CPU contexts from altering the pool while this operation is in progress.

This is equivalent to:

   pmovebetween(p, e, plast(p), NULL);

but is considerably faster.

A common use for this function is to insert an element allocated when the POOL_LINKED_BY_APP attribute is set for the pool, at the end of the allocated list. This is useful when, say, an ISR allocates and uses a buffer without placing it on the allocated list. Only when the buffer is complete does the ISR use this function to place it on the queue for reading by the main application.

The pool MUST be set to being a linked pool by using:

   pool_link(p, <non-zero>);

otherwise the results are undefined.

PARAMETERS
p Pointer to pool handle structure, as previously passed to pool_init().
e Address of element to move. If NULL, then this function behaves as plast().

RETURN VALUE
Same as the “e” parameter, unless “e” is NULL in which case the existing last element is returned as per plast().

LIBRARY
POOL.LIB

SEE ALSO
pmovebetween, pool_link
pputlast_fast

```c
void * pputlast_fast(Pool_t * p, void * e);
```

**DESCRIPTION**

See description under `pputlast()`. This is an assembler-callable version (do not call from C). It does not issue IPSET protection or check parameters.

**Registers:**
- Parameters in IX ("p") and DE ("e")
- Trashes F, DE, BC
- Return value in HL

**PARAMETERS**

- **p**
  - Pointer to pool handle structure, as previously passed to `pool_init()`.
  - Pass in IX register

- **e**
  - Address of element to move. Pass in DE register. If NULL, then this function behaves as `plast_fast()`.

**RETURN VALUE**

In HL. Same as the "e" parameter, unless "e" is NULL in which case the existing last element is returned as per `plast_fast()`.

**LIBRARY**

POOL.LIB

**SEE ALSO**

`pmovebetween`, `pool_link`

premain

```c
void premain( void );
```

**DESCRIPTION**

Dynamic C calls `premain` to start initialization functions such as `VdInit`. The final thing `premain` does is call `main`. This function should never be called by an application program. It is included here for informational purposes only.

**LIBRARY**

PROGRAM.LIB
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 pool_init().
    e      Address of element to move, obtained by e.g., plast(). 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 plast() 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 cata-
           strophic).
    where  The reference element. The element “e” will be inserted before or after this
           element, depending on the options parameter. If e==where, then there is
           no action. If this parameter is null, then the reference element is assumed
           to be the first element (i.e., whatever pfirst() 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:

           POOL_INSERT_BEFORE
           POOL_INSERT_AFTER

           which specifies whether “e” is to be inserted before or after “where.”
preorder (cont’d)

RETURN VALUE

Returns the parameter value “e” unless “e” was null, in which case the value of \texttt{plast()}, when called at function entry, would be returned.

\textbf{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

```c
void * r;
void * s;

s = pnext(p, pfirst(p)); // s is second element
r = plast(p); // r is last element
preorder(p, s, r, \texttt{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, \texttt{POOL\_INSERT\_BEFORE});

// which is identical to:
preorder(p, plast(p), pfirst(p), \texttt{POOL\_INSERT\_BEFORE});
```

LIBRARY

\texttt{POOL.LIB}

SEE ALSO

pool_init, pool_link, plast, pfirst, pnext, pprev, pmovebetween
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

fmt  String to be formatted.
...
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.

As shown in the screenshot above, a value to the right of “.” causes the string to be displayed with that number of characters, ignoring extra characters. A value by itself or to the left of “.” causes padding. Negative values cause the string to be left justified, with spaces added to the right if necessary. Positive values cause the string to be right justified, with spaces added to the left if necessary.
printf (cont’d)

%ls - null terminated string in xmem

This conversion specifier is identical to “%s” but the strings come from extended memory instead of root memory.

```c
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

```c
short n;
n = 30000;
printf("%d", n);  // 30000
printf("%5d", n);  // 30000
printf("%6d", n);  // 30000
printf("%4d", n);  // ****
```

```c
unsigned short n;
n = 40000;
printf("%u", n);  // 40000
```

```c
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

```c
short n;
n = -25536;
printf("%u", n);  // 40000
```

```c
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.

```c
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("%.3f", f); // -88.889
printf("%.3e", f); // -88.889
printf("%.3f", f); // -88.889
printf("%0.3f", f); // -88.889
printf("%.3f", f); // -88.889
printf("%0.3e", f); // -89.001
printf("%.1e", f); // -89.001
printf("%.3e", f); // -89.001
printf("%.3e", f); // -89.001
printf("%.3e", f); // -89.001
printf("%.3e", f); // -89.001
printf("%.3e", f); // -89.001
```
printf (cont’d)

%g - floating point

(Same as %f or %e depending upon value and precision.)

```c
float f, g, h;
f = -888.8888;
g = 888888.0
g = 8888880.0
```

```c
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.

```c
int i, *iptr;
i = 0;
ptr = &i;
```

```c
printf("%p\n",ptr); // prints value of ptr in hex.
```

// logical memory location of i

%lp - pointer

Specifies a 32-bit physical pointer.

```c
long i, *iptr;
i = 0;
ptr = &i;
```

```c
printf("%lp\n",ptr); // prints value of ptr in hex.
```

// physical memory location of i
printf (cont’d)

%x - hexadecimal
Result in lowercase
Width specifier l: short values must not include l; without l, long values are treated as short
Precision specifier n: must be at least as large as total width; treats argument as if it were un-
signed

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**

```c
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**

```c
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**

```c
unsigned long pwm_init( unsigned long frequency );
```

**DESCRIPTION**

Sets the base frequency for the pulse width modulation (PWM) and enables the PWM driver on all four channels. The base frequency is the frequency without pulse spreading. Pulse spreading (see `pwm_set()`) will increase the frequency by a factor of 4.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

**PARAMETER**

- **frequency**  
  Requested frequency (in Hz)

**RETURN VALUE**

The actual frequency that was set. This will be the closest possible match to the requested frequency.

**LIBRARY**

- `PWM.LIB (was in R3000.LIB prior to DC 10)`
pwm_set

int pwm_set( int channel, int duty_cycle, int options );

DESCRIPTION
Sets a duty cycle for one of the pulse width modulation (PWM) channels. The duty cycle can be a value from 0 to 1024, where 0 is logic low the whole time, and 1024 is logic high the whole time. Option flags are used to enable features on an individual PWM channel. Bit masks for these are:

- PWM_SPREAD - sets pulse spreading. The duty cycle is spread over four separate pulses to increase the pulse frequency.
- PWM_OPENDRAIN - sets the PWM output pin to be open-drain instead of a normal push-pull logic output.

This function is intended for use with the Rabbit 3000 and Rabbit 4000.

PARAMETERS
- **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
PWM.LIB (was in R3000.LIB prior to DC 10)
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

xmем long pxalloc_fast( Pool_t * p );

DESCRIPTION

Return next available free element from the given pool. Eventually, your application should re-

 Return this element to the pool using pxfree() to avoid memory leaks.

This is an assembler-only version of pxalloc().

*** Do _not_ call this function from C. ***

pxalloc_fast does not perform any IPSET protection, parameter validation, or update the

high-water mark. pxalloc_fast is a root function. The parameter must be passed in IX, and

the returned element address is in BCDE.

REGISTERS

Parameter in IX

Trashes AF, HL

Return value in BCDE, carry flag.

EXAMPLE

    ld ix,my_pool
    lcall pxalloc_fast
    jr c,.no_free
; BCDE points to element

PARAMETERS

p

Pool handle structure, as previously passed to pool_init()  Pass this

in the IX register.

RETURN VALUE

C flag set: No free elements are available. (BCDE is undefined in this case.)

NC flag: BCDE points to an element If the pool is not linked, your application must not write

more than p->elsize bytes to it (this was the elsize parameter passed to

pool xinit()). If the pool is linked, you can write (p->elsize-8) bytes to it. (An ele-

ment has 8 bytes of overhead when the pool is linked.)

LIBRARY

POOL.LIB

SEE ALSO

pool_init, pfree_fast, pavail_fast, 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 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
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, pxfree, pxlast, pxnext, pxprev
pxfirst_fast

xmem long pxfirst_fast( Pool_t * p );

DESCRIPTION

Get the first allocated element in an xmem pool. The pool MUST be set to being a linked pool using pool_link(p, <non-zero>); otherwise, the results are undefined.

This is an assembler-only version of pxfirst().

*** Do _not_ call this function from C. ***

Registers

Parameter in IX
Trashes F, HL
Return value in BCDE, carry flag

Example

    ld ix,my_pool
    lcall pxfirst_fast
    jr c,.no elems
    ; BCDE points to first element

PARAMETERS

p

Pool handle structure, as previously passed to pool_init(). Pass this in IX register.

RETURN VALUE

C flag set: There are no allocated elements
C flag clear (NC): BCDE points to first element

LIBRARY

POOL.LIB

SEE ALSO

pool_xinit, pool_link, pxfirst, pxnext_fast
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 pxalloc().
e            Element to free, which was returned from pxalloc().

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

```c
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**

```c
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 `palloc()` or `palloc_fast`. This must be in the IX register.

- **e**
  
  Element to free, which was returned from `palloc()`. 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`
long pxlast( Pool_t * p );

DESCRIPTION
Get the last allocated element in an xmem pool. The pool MUST be set to being a linked pool using pool_link(p, <non-zero>); otherwise, the results are undefined.

PARAMETERS
p
Pool handle structure, as previously passed to pool_xinit().

RETURN VALUE
0: There are no allocated elements
!0: Pointer to last, i.e., youngest, allocated element

LIBRARY
POOL.LIB

SEE ALSO
pool_xinit, pool_link, pxalloc, pxfree, pxfirst
pxlast_fast

xmem long pxlast_fast( Pool_t * p );

DESCRIPTION
Get the last allocated element in an xmem pool. The pool MUST be set to being a linked pool using pool_link(p, <non-zero>); otherwise, the results are undefined.

This is an assembler-only version of pxlast().

*** Do _not_ call this function from C. ***

Registers
Parameter in IX
Trashes F, HL
Return value in BCDE, carry flag

Example

     ld ix, my_pool
     lcall pxlast_fast
     jr c,.no elems
 ;BCDE points to last element

PARAMETERS

p          Pool handle structure, as previously passed to pool_xinit(). Pass this in IX register.

RETURN VALUE
C flag set: There are no more elements
C flag clear (NC): BCDE points to last element

LIBRARY
POOL.LIB

SEE ALSO
pool_xinit, pool_link, pxlast, pxprev_fast
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:

```c
long e;
Pool_t * p;
for (e = pxfirst(p); e; e = pxnext(p, e)) {
    ...
}
```

PARAMETERS

- **p**
  - Pool handle structure, as previously passed to pool_xinit().

- **e**
  - Previous element address, obtained by e.g. pxfirst(). This must be an allocated element in the given pool, otherwise the results are undefined. Be careful when iterating through a list and deleting elements using pxfree(): once the element is deleted, is is no longer valid to pass its address to this function. If this parameter is zero, then the result is the same as pxfirst(). This ensures the invariant pxnext(p, pxprev(p, e)) == e.

RETURN VALUE

- 0: There are no more elements
- !0: Pointer to the next allocated element

LIBRARY

POOL.LIB

SEE ALSO

pool_xinit, pool_link, pxalloc, pxfree, pxfirst, pxprev
**pxnext_fast**

```c
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**

```asm
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 `pool_xinit()`. Pass this in the IX register.

- **e**  
  Current element, address in BCDE register. See `pxnext()` 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`
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 pool_xinit().
e Previous element address, obtained by e.g., pxlast(). This must be an
allocated element in the given pool; otherwise, the results are undefined.
Be careful when iterating through a list and deleting elements using
pxfree(): once the element is deleted, it is no longer valid to pass its ad-
dress to this function. If this parameter is zero, then the result is the same
as pxlast(). 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**

```c
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**

```assembly
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 `pool_xinit()`. Pass this in IX register.

- **e**
  - Current element, address in BCDE register. See `pxprev()` 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

```c
char qd_error( int channel );
```

**DESCRIPTION**

Gets the current error bits for that qd channel. This function is intended to be used with the Rabbit 3000 and Rabbit 4000.

**PARAMETERS**

- **channel**: The channel to read errors from (currently 1 or 2).

**RETURN VALUE**

Set of error flags, that can be decoded with the following masks:

- QD_OVERFLOW 0x01
- QD_UNDERFLOW 0x02

**LIBRARY**

- QD.LIB (was in R3000.LIB prior to DC 10)
**qd_init**

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

**DESCRIPTION**

If your board has a Rabbit 3000A microprocessor installed, the quadrature decoder can be set for 10 bit counter operation. For 10 bit operation, add the following macro at the top of your application program.

```c
#define QD_10BIT_OPERATION
```

If the above macro is not defined then the quadrature decoder defaults to 8 bit counter operation. With the Rabbit 3000 processor you must use the default 8-bit operation; defining the 10-bit macro will cause a compile time error.

Sample program `Samples/Rabbit3000/QD_Phase_10bit.c` demonstrates the use of the macro.

If your board has a Rabbit 4000 microprocessor installed, the quadrature decoder inputs must be chosen with one of the following defines. Define only one per quadrature decoder.

```c
#define QD1_USEPORTD     // use port D pins 1 and 0
#define QD1_USEPORTEL    // use port E pins 1 and 0
#define QD1_USEPORTEH    // use port E pins 5 and 4

#define QD2_USEPORTD     // use port D pins 3 and 2
#define QD2_USEPORTEL    // use port E pins 3 and 2
#define QD2_USEPORTEH    // use port E pins 7 and 6
```

If no macro is defined for a decoder, that decoder will be disabled.

**PARAMETERS**

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

**LIBRARY**

- `QD.LIB` (was in R3000.LIB prior to DC 10)
qd_read

long qd_read( int channel );

DESCRIPTION

Reads the current quadrature decoder count. Since this function waits for a clear reading, it can potentially block if there is enough flutter in the decoder count.

This function is intended to be used with the Rabbit 3000 and Rabbit 4000.

PARAMETERS

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

RETURN VALUE

Returns a signed long for the current count.

LIBRARY

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

qd_zero

void qd_zero( int channel );

DESCRIPTION

Sets the count for a channel to 0. This function is intended to be used with the Rabbit 3000 and Rabbit 4000.

PARAMETERS

channel The channel to reset (currently 1 or 2)

LIBRARY

QD.LIB (was in R3000.LIB prior to DC 10)
**qsort**

```c
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 `strcmp(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 `strcmp()` — 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.

```c
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 ≤ v < 1.0. Uses algorithm:

\[ \text{rand} = ( 5 \times \text{rand} ) \mod 2^{32} \]

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 ≤ v < 1.0.

LIBRARY
MATH.LIB

SEE ALSO
randb, randg, srand
**randb**

`float randb( void );`

**DESCRIPTION**

Uses algorithm:

\[
\text{rand} = ( 5 \times \text{rand} ) \mod 2^{32}
\]

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

```c
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
int RdPortI( int port );

DESCRIPTION

Reads an internal I/O port specified by the argument (use RdPortE() for external port).

All of the Rabbit internal registers have predefined macros corresponding to the name of the
register. PADR is #defined to be 0x30, etc.

PARAMETERS

port Address of internal I/O port

RETURN VALUE

Returns an integer, the lower 8 bits of which contain the result of reading the port specified by the argument. Upper byte contains zero.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortE, BitRdPortI, WrPortI, BitWrPortI, BitRdPortE, WrPortE, BitWrPortE
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

```c
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

```c
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

```c
int readUserBlock( void * dest, unsigned addr, unsigned numbytes );
```

## DESCRIPTION

Reads a number of bytes from the User block on the primary flash to a buffer in root memory. Please note that portions of the User block may be used by the BIOS for your board to store values. For example, any board with an A to D converter will require the BIOS to write calibration constants to the User block. For some versions of the BL2000 and the BL2100 this memory area is 0x1C00 to 0x1FFF. See the user’s manual for your particular board for more information before overwriting any part of the User block. Also, see the *Rabbit Microprocessor Designer’s Handbook* for more information on the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), `readUserBlockArray()` should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using µC/OS-II and `SPI_USE_UCOS_MUTEX` is defined, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error `ERR_SPI_MUTEX_ERROR` will occur. See the description for `rcm43_InitUCOSMutex()` for more information on using µC/OS-II and `SPI_USE_UCOS_MUTEX`.

### 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
- -2: No valid ID block found (block version 3 or later)

The return values below are applicable only if `SPI_USE_UCOS_MUTEX` is not defined:

- -ETIME: (Serial flash only, time out waiting for SPI)
- positive N: (Serial flash only, SPI in use by device N)

### LIBRARY

IDBLOCK.LIB

### SEE ALSO

- `writeUserBlock`
- `readUserBlockArray`
**readUserBlockArray**

```c
int readUserBlockArray( void * dests[], unsigned numbytes[], int numdests, unsigned addr );
```

**DESCRIPTION**

Reads a number of bytes from the User block on the primary flash to a set of buffers in root memory. This function is usually used as the inverse function of `writeUserBlockArray()`.

This function was introduced in Dynamic C version 7.30.

**Note:** Portions of the User block may be used by the BIOS to store values such as calibration constants. See the manual for your particular board for more information before overwriting any part of the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), `readUserBlockArray()` should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using µC/OS-II and `_SPI_USE_UCOS_MUTEX` is #defined, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error `ERR_SPI_MUTEX_ERROR` will occur. See the description for `_rcm43_InitUCOSMutex()` for more information on using µC/OS-II and `_SPI_USE_UCOS_MUTEX`.

**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)

The return values below are applicable only if `_SPI_USE_UCOS_MUTEX` is not #defined:

- `ETIME`: (Serial flash only, time out waiting for SPI)
- postive N: (Serial flash only, SPI in use by device N)

**LIBRARY**

```
IDBLOCK.LIB
```

**SEE ALSO**

- `writeUserBlockArray`, `readUserBlock`
**registry_enumerate**

```c
int registry_enumerate( RegistryContext * r, int (*f)(), int keyvalues, void far * ptr);
```

**DESCRIPTION**

Enumerate registry `r->old_spec`, calling the specified function “f” for each section header and, optionally, key=value pair.

The `registry_get()` function also performs enumeration; in fact it is a wrapper for this function.

**PARAMETERS**

- **r**
  - RegistryContext structure, with at least the `old_spec` field initialized.
  - For example, use `registry_prep_read()` to set up the struct correctly.
  - `r->old_spec`: Open resource handle of a readable resource containing the registry settings. This is read from the current seek position, thus in most cases call this function with a freshly opened resource handle.

- **f**
  - Callback function to be invoked. The function prototype must be as follows:
    ```c
    int f(void far * ptr,
          int new_sect,
          char * sect,
          char far * key,
          char far * value) { ... }
    ```
  - where the parameters are:
    - `ptr` - this is passed through from the 4th parameter to the `registry_enumerate()` function (see below).
    - `new_sect` - boolean indicating whether this call is to introduce a new section. If true, then 'sect' is the new section name, and 'key' and 'value' are not relevant.
    - `sect` - name of section if `new_sect` flag is true
    - `key` - key (field) ascii string if `new_sect` is false
    - `value` - value as an ascii string if `new_sect` is false.

- **keyvalues**
  - Boolean indicating whether the callback function is to be invoked for key=value pairs (if true). In either case, the callback is invoked whenever a new section is found, and the `new_sect` callback parameter will be set true.

- **ptr**
  - An arbitrary pointer which will be passed through to the callback on each invocation.
registry Enumerate (cont’d)

RETURN VALUE

<0: failure to write or read the resource
0: success

LIBRARY

registry.lib

SEE ALSO

sspec_open, registry_read, registry_update, registry_get,
registry_prep_read, registry_finish_read
```c
int registry_get( char * basename, char far * section,
                 RegistryEntry * re, ServerContext * sctx, int (*f)(),
                 int keyvalues, void far * ptr);
```

**DESCRIPTION**

Convenience function for reading and/or enumerating registry contents. This basically combines calls to the following functions:

- `registry_prep_read()`
- `registry_read()` and/or `registry_enumerate()`
- `registry_finish_read()`

If the field array (re) is not NULL, then `registry_read()` will be called. If the callback function (f) is not NULL, then `registry_enumerate()` will be called. If both re and f are not NULL, then read will be invoked before enumerate.

**PARAMETERS**

- **basename**: Base name of registry file, as a Zserver resource name. This file must not have an extension, since the extensions ".1", ".2" and so on are appended to the name.
- **section**: Section name to read (may be NULL to read the anonymous section at the start of the registry file).
- **re**: Array of fields to read. See `registry_read()` function description for details.
- **sctx**: Server context.
- **f()**: Callback function. See `registry_enumerate()` for details.
- **keyvalues**: Boolean indicating whether callback receives key=value pairs as well as section headers. If false, it only receives section headers.
- **ptr**: Arbitrary application data which will be dutifully passed through to the callback without alteration.

**RETURN VALUE**

- `<0`: general failure, code will be negative of one of the codes in `ERRNO.LIB`.
- `0`: OK

**LIBRARY**

`register.lib`

**SEE ALSO**

`registry_prep_read`, `registry_read`, `registry_finish_read`, `registry_enumerate`, `registry_update`
**registry_finish_read**

```c
int registry_finish_read( RegistryContext * r);
```

**DESCRIPTION**

Finish reading a registry, and clean up resources. Most applications will use the sequence of functions:

```c
registry_prep_read()
registry_read() and/or registry Enumerate()
registry_finish_read()
```

**PARAMETER**

`r` RegistryContext struct, as set by `registry_prep_read()`.

**RETURN VALUE**

- `<0`: general failure, code will be negative of one of the codes in `ERRNO.LIB`.
- `0`: OK.

**LIBRARY**

`registry.lib`

**SEE ALSO**

`registry_read`, `registry_prep_read`, `registry_prep_write`, `registry_write`, `registry_finish_write`, `registryEnumerate`, `registry_update`, `registry_get`
**registry_finish_write**

```c
int registry_finish_write( RegistryContext * r);
```

**DESCRIPTION**

Finish updating a registry, and clean up resources. Most applications will use the sequence of functions

```c
registry_prep_write()
registry_write()
registry_finish_write()
```

**PARAMETER**

- `r` RegistryContext structure, as set by `registry_prep_read()`.

**RETURN VALUE**

- `<0: general failure, code will be negative of one of the codes in ERRNO.LIB.
- `0: OK`

**LIBRARY**

registry.lib

**SEE ALSO**

- `registry_read`, `registry_prep_read`, `registry_prep_write`,
- `registry_write`, `registry_finish_read`, `registryEnumerate`,
- `registryUpdate`, `registryGet`
int registry_prep_read( RegistryContext * r, char * basename, ServerContext * context);

DESCRIPTION

Prepare for reading a registry. This function helps organize registry resources in order to create a robust registry.

Most applications will use the sequence of functions:

registry_prep_read()
registry_read() and/or registryEnumerate()
registryFinish_read()

or simply
registry_get()

Registry updates require reading from an old registry, editing it, then writing the modified result to a new registry resource. This requires two resources to be open. Normally, the "old" registry will be deleted once the update is successful. If there is a power outage or reset during this process, it is possible for two registry files to exist when the system is restarted. This causes problems, since one of the registries may be corrupt. This API imposes a naming convention on the old/new resources so that a non-corrupt registry can always be found.

The algorithm used appends an extension to the basename resource name. The extension is ".1", ".2" or ".3". The "current" registry resource will cycle through these extensions. It is assumed that exactly 0, 1 or 2 of these resources will exist at any time. This means that at least one of the possible resource names will not exist. (If all three exist, then the behavior is undefined, since the resources must have been created outside the registry system. The application is responsible for ensuring this does not happen, otherwise the ability to find a non-corrupt registry will be compromised).
If none of the resources exist, then this indicates a brand new registry. If exactly one exists, then this is the old (and presumed non-corrupt) registry. If two exist, it is assumed that one of the resources is OK and the other corrupt. Since there are only 3 possible extensions, and they increment in wrap-around fashion, the “lowest” numbered extension is assumed to be the non-corrupt one, with “lowest” being in the sense of modulo 3. This is summarized in the following table:

<table>
<thead>
<tr>
<th>Existing Extensions</th>
<th>Assumed Non-corrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>None, new registry</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1,2</td>
<td>1 (2 will be deleted)</td>
</tr>
<tr>
<td>2,3</td>
<td>2 (3 will be deleted)</td>
</tr>
<tr>
<td>1,3</td>
<td>3 (1 will be deleted)</td>
</tr>
<tr>
<td>1,2,3</td>
<td>Should not happen - will arbitrarily pick 1 and delete 2,3.</td>
</tr>
</tbody>
</table>

In the case that more than one registry extension was found, the presumed corrupt resource is automatically deleted to clean up the registry.

PARAMETERS

- **r**: RegistryContext structure. This is used to pass information in a consistent manner between the major registry API functions. It may be passed uninitialized to this function. This function fills in the r->old_spec field to indicate the open resource which will be used by `registry_read()`. The value may also be set to -1 if there was an error or no existing resource could be located.

- **basename**: Base name (including path) of the registry. This should NOT include any extension (e.g. ".foo") since the extension is manipulated by this function. In practice, this will need to be a resource name on non-volatile storage, which supports names with extensions. In practice, this limits the appropriate filesystem to FAT filesystem only. For example

  ```
  registry_prep_read("/A/myreg", &spec);
  ```

  will select from a set of registry files called /A/myreg.1, /A/myreg.2, /A/myreg.3 of which, normally, only one will exist at any time.
**registry_prep_read (cont’d)**

**context**    ServerContext struct. E.g. from http_getContext().

**RETURN VALUE**

- `<0`: General failure, code will be negative of one of the codes in ERRNO.LIB.
- `0`: there is currently no resource of the given name. This is not necessarily an error, since it will be returned if the registry has not yet been created.
- `1, 2, 3`: An existing presumed non-corrupt resource has been opened. The numeric return code indicates which of the extensions was located.

**LIBRARY**

*register.lib*

**SEE ALSO**

registry_read, registry_finish_read, registry_prep_write, registry_write, registry_finish_write, registryEnumerate, registry_update, registry_get
```c
int registry_prep_write( RegistryContext * r, char * basename,
                        ServerContext * context);
```

**DESCRIPTION**

Prepare for updating a registry. This function helps organize registry resources in order to create a robust registry.

Most applications will use the sequence of functions

```c
    registry_prep_write()
    registry_write()
    registry_finish_write()
```

or, more simply, just

```c
    registry_update()
```

See the function description for `registry_prep_read()` for details concerning the organization of registry files.

Like `registry_prep_read()`, this function opens an existing presumed non-corrupt registry for reading, and also a new empty registry (the "next" registry) for writing the updated results, as required by `registry_write()`.

**PARAMETERS**

- **r**
  RegistryContext struct. This is used to pass information in a consistent manner between the major registry API functions. It may be passed uninitialized to this function.

- **basename**
  Base name (including path) of the registry. This should NOT include any extension (e.g. ".foo") since the extension is manipulated by this function.
  In practice, this will need to be a resource name on non-volatile storage, which supports names with extensions. In practice, this limits the appropriate filesystem to FAT filesystem only. For example
  ```c
  registry_prep_write("/A/myreg", &oldspec, &newspec);
  ```
  will select from a set of registry files called
  `/A/myreg.1, /A/myreg.2, /A/myreg.3` of which, normally, only two will exist at any time; one will be opened for reading, and the other will be empty and ready for writing.

- **context**
  ServerContext structure. E.g. from `http_getContext()`.
registry_prep_write (cont’d)

RETURN VALUE

<0: general failure, code will be negative of one of the codes in ERRNO.LIB.

0: there is currently no resource of the given name. *oldp will be set to -1 in this case. This is not necessarily an error, since it will be returned if the registry has not yet been created. You can pass *oldp to registry_write() in this case, and it will correctly create the new registry without attempting to read the (non-existent) "old" registry.

1, 2, 3: An existing presumed non-corrupt resource has been opened, and the open resource handle returned in *oldp. The numeric return code indicates which of the extensions was located. Note that the "new" registry file will be this number plus 1 (except that 4 becomes 1).

LIBRARY

register.lib

SEE ALSO

registry_read, registry_finish_read, registry_prep_read, registry_write, registry_finish_write, registry.enumerate, registry_update, registry_get
**registry_read**

```c
int registry_read( RegistryContext * r, char far * section,
                  RegistryEntry far * entries);
```

**DESCRIPTION**

Read the registry r->old_spec using the specified registry entries. Only entries in the named “section” are read, and the results are placed at the locations pointed to by the RegistryEntry array elements.

**Note:** Since this function requires some temporary malloc memory, you should ensure that there is at least _REGBUF_SIZE bytes of available system-space malloc memory. The _REGBUF_SIZE macro defaults to 1025 bytes, but you may override this definition before #use registry.lib.

**r** RegistryContext structure, with at least the old_spec field initialized. For example, use registry_prep_read() to set up this structure correctly.

- `r->old_spec:`
  - Open resource handle of a readable resource containing the registry settings. This is read from the current seek position, thus in most cases you will want to call this function with a freshly opened resource handle.

**section** Section name. If NULL or empty string, then the first (anonymous) section of the registry is implied.

**entries** List of registry entries to read. See the registry_write() description for details. The “value” field will be set to point to the location where the read value is stored. If the key does not exist in the specified section, then the contents at this location will be untouched. Thus, you can set “default” values at each location before calling registry_read().

As for registry_write(), the list MUST be terminated with an entry with the REGOPTION_EOL option.

**RETURN VALUE**

- `<0: failure to write or read the resource`
- `0: success`

**LIBRARY**

register.lib

**SEE ALSO**

sspec_open, registry_write, registry_update, registry_get, registry_prep_read, registry_finish_read
int registry_update( char * basename, char far * section, 
    RegistryEntry * re, ServerContext * sctx);

DESCRIPTION
Convenience function for updating a registry with a minimum of fuss. Basically combines the 
function calls:

    registry_prep_write()
    registry_write()
    registry_finish_write()

PARAMETERS

    basename  Base name of registry file, as a Zserver resource name. This file must not 
              have an extension, since the extensions ".1", ".2" and so on are appended 
              to the name.
    section   Section name to update (may be NULL to update the anonymous section 
              at the start of the registry file).
    re        Array of update commands. See the registry_write() function de- 
              scription for details. If this pointer is NULL, the entire section is deleted.
    sctx      Server context.

RETURN VALUE
<0: general failure, code will be negative of one of the codes in ERRNO.LIB.
  0: OK

LIBRARY
    register.lib

SEE ALSO
    registry_prep_write, registry_write, registry_finish_write, 
    registry_get
**registry_write**

```c
int registry_write( RegistryContext * r, char far * section,
                   RegistryEntry far * entries);
```

**DESCRIPTION**

Modify the old registry `r->old_spec` using the specified registry entries, writing the result to `r->new_spec`. Only entries in the named “section” may be altered. This function also allows entries and sections to be deleted.

The new and old files must be different, since this function depends on reading from the old file, performing the requested modifications, and writing the new file -- this is all done line-by-line. Generally, you will need two resource files which will alternate. Only when the modifications are successfully complete will the old file be deleted. This makes the update process more resistant to corruption caused by e.g., the user turning off the power in the middle of the update.

The helper function `registry_prep_write()` automates this process. The function `registry_update()` encapsulates the basic registry update process.

**NOTE:** since this function requires some temporary malloc memory, you should ensure that there is at least `_REGBUF_SIZE` bytes of available system-space malloc memory. The `_REGBUF_SIZE` macro defaults to 1025 bytes, but you may override this definition before #use registry.lib.

Registry resources are similar to Windows “.ini” file format. They are ASCII formatted (and thus human readable) and consist of one or more “sections,” each of which has zero or more key=value lines. For example:

```ini
[net settings]
ip=10.10.6.100
ssid=Rabbit
[app settings]
some integer=23
a string=hello world
```

Each section is headed by a string enclosed in square brackets. Within each section is a list of key strings followed by '=' followed by the value of that entry. The key string is arbitrary except that it cannot start with '[' or contain any '=' or null or newline characters. The value string is arbitrary except that newline and null characters are not allowed. Section names are arbitrary except they cannot contain ']' or null or newline characters. Spaces are always significant. In particular, don't put spaces on either side of the '=' separator.

If there are duplicate keys in the entries table, then it is undefined which of the entries actually gets stored. Don't do it.

Normally, you do not need to be concerned with the above format rules, since the library functions enforce them.
If you need to store null (binary zero) or newline (binary 0x0A or, in C syntax, "\n") then your application will need to use some sort of convention for escaping such characters, or you can use the \texttt{REGOPTION\_BIN()} option which will store the string expanded into ASCII hexadecimal, which is completely safe.

Individual key/value entries may be deleted by specifying the \texttt{REGOPTION\_DELETE} flag with the appropriate entries.

\textbf{PARAMETERS}

\begin{itemize}
\item \texttt{r} RegistryContext structure, with at least the \texttt{old\_spec} and \texttt{new\_spec} fields initialized. For example, use \texttt{registry\_prep\_write()} to set up this structure correctly.
\item \texttt{r->old\_spec}: Open resource handle of a readable resource containing the old registry settings. This is read from the current seek position, thus in most cases you will want to call this function with a freshly opened resource handle. This may also be -1, which indicates there is *no* old registry to update, and a new registry will be written to \texttt{new\_spec}.
\item \texttt{r->new\_spec}: Open resource handle of a writable resource, to which the old registry (modified with the given settings) will be written. Normally, this should initially be an empty resource file. The new settings will be written starting at the current seek position in this resource.
\end{itemize}

Note that the resource handles remain open when this function returns.

\item \texttt{section} Section name. If NULL or empty string, then the first (anonymous) section of the registry is implied.
entries List of replacement registry entries. The list MUST be terminated with an entry with the REGOPTION_EOL option.

Caution: If this pointer is NULL, then the entire section is deleted.

Each element in this array is as follows:

typedef struct {
    char far * key; // Entry key. Must not contain '=' or newlines, and must not start with ']. Must be null-terminated.
    void far * value; // Entry value. Type determined by options. If the REGOPTION_STRING option is set, this must not contain newlines and must be null terminated.
    int options; // Entry options and flags: If value is greater than zero, then value is an arbitrary binary value with the specified length. It will be stored in the registry with twice that many ascii hex digits. If value is <= -10, then it is an ascii string with max length of (-options-8) ascii string with max length of (-options-8) Otherwise, this field is a simple enumeration indicating the data type as follows:

#define REGOPTION_EOL 0 // End of list
#define REGOPTION_SHORT (-1) // Signed short (2 byte) - stored as decimal
#define REGOPTION_LONG (-2) // Signed long (4 byte) - stored as decimal
#define REGOPTION_BOOL (-3) // int (2 byte) - stored as 1 (if non-zero) or 0
#define REGOPTION_FLOAT (-4) // IEEE float (4 byte)
    // Only avail if STDIO_DISABLE_FLOATS *not* defined, stored in %f format
#define REGOPTION_RESV5 (-5)
#define REGOPTION_RESV6 (-6)
#define REGOPTION_DELETE (-7) // Delete this entry if found
#define REGOPTION_NOP (-8) // No operation: convenience for constructingRegistryEntry lists.
#define REGOPTION_RESV9 (-9) // For variable length data...

#define REGOPTION_BIN(len) (len)
    // Binary of given fixed length - stored expanded into ascii hexadecimal.
    // len must be 1.._REGBUF_SIZE/2-M where M is the size of the key plus 2.
    // As a rule of thumb, be careful when len is more than about 256.
#define REGOPTION_STRING(len) (-8-(len))
    // Null-terminated string up to len chars counting the null terminator - stored as-is.
    // len must be at least 2. len must not be more than _REGBUF_SIZE-M where M is the size of the key plus 2. As a rule of thumb, be careful when len is more than about 512.

word work; // Work field for registry read/write lib functions
    // May be left uninitialized by the caller unless otherwise noted in the function description.
} RegistryEntry;
registry_write (cont’d)

RETURN VALUE

<0: failure to write or read the resource
0: success

LIBRARY

REGISTRY.LIB

SEE ALSO

sspec_open, registry_read, registry_update, registry_get,
registry_prep_write, registry_finish_write
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:

\[ *(\text{long} *)\text{address} \&= ~(1L << \text{bit}) \]

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

---

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:

\[ *(\text{long} *)\text{address} \&= ~(1L << \text{bit}) \]

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

```c
void ResetErrorLog( void );
```

**DESCRIPTION**

This function resets the exception and restart type counts in the error log buffer header. This function is not called by default from anywhere. It should be used to initialize the error log when a board is programmed by means other than Dynamic C, cloning, the Rabbit Field Utility (RFU), or a service processor. For example, if boards are mass-produced with pre-programmed flash chips, then the test program that runs on the boards should call this function.

**LIBRARY**

`ERRORS.LIB`

---

root2vram

```c
int root2vram( void * src, int start, int length );
```

**DESCRIPTION**

This function copies data to the VBAT RAM. Tamper detection on the Rabbit 4000 erases the VBAT RAM with any attempt to enter bootstrap mode.

**PARAMETERS**

- `src` The address to the data in root to be copied to vbat ram.
- `start` The start location within the VBAT RAM (0-31).
- `length` The length of data to write to VBAT RAM. The length should be greater than 0.

The parameters `length + start` should not exceed 32.

**LIBRARY**

`VBAT.LIB`

**SEE ALSO**

`vram2root`
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
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 TZNAME macro. If TZNAME 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**

```c
void runwatch( void );
```

**DESCRIPTION**

Runs and updates watch expressions if Dynamic C has requested it with a Ctrl-U. Should be called periodically in user program.

**LIBRARY**

SYS.LIB

---

**sdspi_debounce**

```c
int sdspi_debounce( sd_device * sd );
```

**DESCRIPTION**

This function waits for and debounces the card insertion switch. When it returns True (1), then a card is fully inserted.

**PARAMETER**

- `sd` The device structure for the SD card.

**RETURN VALUE**

- 1: Success, card fully inserted
- 0: No card present

**LIBRARY**

SDFLASH.LIB
int sdspi_get_csd(sd_device * sd);

DESCRIPTION
This function is called to execute protocol command 9 to retrieve the SD card's Card Specific Data (CSD) and store it in the respective SD driver configuration object. The CSD data is used to determine the SD card's physical storage and timing attributes.

PARAMETERS
sd The device structure for the SD card.

RETURN VALUE
0: Success
-EIO: I/O error
-EINVAL: Invalid parameter given
-ENOMEDIUM: No SD card in socket
-ESHAREDBUSY: Shared SPI port busy

LIBRARY
SDFLASH.LIB
int sdspi_get_scr( sd_device * sd );

DESCRIPTION

This function executes application specific command 51 to retrieve the SD card's Configuration Register (SCR) and store it in the respective SD driver configuration object. The SCR data is used to identify the SD card's physical interface version and security version. It also contains erase state (all 0's or 1's) and supported bus widths.

PARAMETERS

sd The device structure for the SD card.

RETURN VALUE

0: Success
-EINVAL: Invalid parameter given
-ENOMEDIUM: No SD card in socket
-ESHARED: Shared SPI port busy

LIBRARY

SDFLASH.LIB
### sdspi_getSectorCount

```c
long sdspi_getSectorCount( sd_device * dev );
```

**DESCRIPTION**

Return number of usable 512 byte sectors on an SD card.

**PARAMETER**

- `dev` Pointer to `sd_device` struct for initialized flash device.

**RETURN VALUE**

Number of sectors

**LIBRARY**

`SDFLASH.LIB`

---

### sdspi_get_status_reg

```c
int sdspi_get_status_reg( sd_device *sd, int * status );
```

**DESCRIPTION**

This function is called to execute protocol command 13 to retrieve the status register value of the SD card.

**PARAMETERS**

- `sd` Pointer to the device structure for the SD card.
- `status` Pointer to variable that returns the status.

**RETURN VALUE**

- 0: Success, Card status placed in status
- `EIO`: I/O error
- `ENOMEDIUM`: No SD card in socket
- `ESHAREDBUSY`: Shared SPI port busy

**LIBRARY**

`SDFLASH.LIB`
int sdspi_init_card( sd_device * sd );

DESCRIPTION

Initializes the SD card pointed to by sd. Function executes protocol command “1” which clears HCS bit and activates the card’s initialization sequence.

PARAMETERS

sd Pointer to sd_device structure for the SD card.

RETURN VALUE

0: Success
-EIO: I/O error
-EINVAL: Invalid parameter given
-ENOMEM: No SD card in socket
-ESHAREDBUSY: Shared SPI port busy

LIBRARY

SDFLASH.LIB
**sdspi_initDevice**

```c
int sdspi_initDevice( int indx, sd_dev_interface * sd_dev );
```

**DESCRIPTION**

Initializes the SD card pointed to by `sd_dev` and adds information about the cards interface to the SD device array in the position pointed to by `indx`. Sets up the default block size of 512 bytes used by sector read/write functions. This function should be called before any calls to other sdspi functions.

**PARAMETERS**

- `indx`   Index into the SD device array to add the card.
- `sd_dev` Pointer to `sd_dev_interface` for the SD card.

**RETURN VALUE**

- 0: Success
- -EIO: I/O error
- -EINVAL: Invalid parameter given
- -ENOMEM: No SD card in socket
- -ESHAREDBUSY: SPI port busy

**LIBRARY**

`SDFLASH.LIB`
### sdspi_isWriting

```c
int sdspi_isWriting( sd_device * dev );
```

**DESCRIPTION**

Returns 1 if the SD card is busy writing a sector.

**PARAMETER**

*dev*  
Pointer to initialized sd_device structure for the flash chip

**RETURN VALUE**

1: Busy  
0: Ready, not currently writing

**LIBRARY**

SDFLASH.LIB

### sdspi_notbusy

```c
int sdspi_notbusy( int port );
```

**DESCRIPTION**

This function tests for a busy status from the SD card on the port given. It is assumed that the card is already enabled.

**PARAMETER**

*port*  
The base address for the SD card's SPI port

**RETURN VALUE**

1: The card is not busy, write/erase has ended  
0: The card is busy, write/erase in progress

**LIBRARY**

SDFLASH.LIB
void sdspi_print_dev( sd_device * dev );

DESCRIPTION

  Prints parameters from the SD device structure.

PARAMETER

  dev  Pointer to sd_device structure of the SD card.

LIBRARY

  SDFLASH.LIB
int sdspi_process_command( sd_device *sd, SD_CMD_REPLY * cmd_reply, int mode );

DESCRIPTION

This function sends the command placed in the cmd_reply structure and retrieves a reply and data (optional) as defined in the cmd_reply structure. Pointers to TX and RX buffers are retrieved from the cmd_reply structure and used for command transmission and reply/data reception. Reply is parsed and placed in cmd_reply.reply. Errors encountered will give a negative return value.

The SPI semaphore is obtained before the command is sent. The mode parameter controls whether the semaphore will be released after command execution and reply/data reception. If mode is zero, both semaphore and chip select are active on a successful return. An end command sequence and release of the semaphore must be handled by caller.

If mode is not 0, the semaphore will be released before returning. In addition, if mode is 2 then an SD card reset is in progress. This enables the distinguishing of certain I/O error conditions that would normally be grouped with the -EIO error code and instead return the -EAGAIN error code, indicating reset retries should continue.

PARAMETER

sd Pointer to sd_device structure of the SD card.

cmd_reply Pointer to cmd_reply structure, which contains:

- cmd - command to be executed
- argument - arguments for the command
- reply - storage for command reply
- reply_size - size in bytes of expected reply
- data_size - size in bytes of expected data
- tx_buffer - pointer to TX buffer to use
- rx_buffer - pointer to RX buffer to use

mode One of the following:

- 0 = SPI port semaphore should be retained.
- 1 = If SPI port to be released before return.
- 2 = Attempting SD card reset, otherwise same as mode “1”.
  (Enables -EAGAIN return value.)
### sdspi_process_command (cont’d)

**RETURN VALUE**

0: Success  
- EIO: I/O error  
- EAGAIN: Allowable I/O error during card reset  
- EINVAL: Invalid parameter given  
- ENOMEDIUM: No SD card in socket  
- ESHAREDUSBUSY: Shared SPI port busy

**LIBRARY**

SDFLASH.LIB

### sdspi_read_sector

```c
int sdspi_read_sector( sd_device * sd, unsigned long sector_number,  
            void * data_buffer );
```

**DESCRIPTION**

This function is called to execute protocol command 17 to read a 512 byte block of data from the SD card.

**PARAMETER**

- **sd**: Pointer to `sd_device` structure of the SD card.  
- **sector_number**: The sector number to read.  
- **data_buffer**: Pointer to a buffer for the 512 bytes read.

**RETURN VALUE**

0: Success  
- EIO: I/O error  
- EINVAL: Invalid parameter given  
- ENOMEDIUM: No SD card in socket  
- ESHAREDUSBUSY: Shared SPI port busy

**LIBRARY**

SDFLASH.LIB
int sdspi_reset_card( sd_device * sd );

DESCRIPTION
Resets the SD card pointed to by sd. Function executes protocol command 0 to force the card to Idle mode. This command is sent multiple times to reset the SD card.

PARAMETER
sd Pointer to sd_device structure of the SD card.

RETURN VALUE
0: Success
-EIO: I/O error
-EINVAL: Invalid parameter given
-ENOMEM: No SD card in socket
-ESHAREDBUSY: Shared SPI port busy

LIBRARY
SDFLASH.LIB
sdspi_sendingAP

int sdspi_sendingAP( sd_device * sd );

DESCRIPTION
Sends AP command 55 to set Alternate Command mode on the next command sent to the card. This function does not release the port sharing semaphore unless an error is encountered.

PARAMETER
sd Pointer to sd_device structure of the SD card.

RETURN VALUE
0: Success
-ENOENT: I/O error
-ENOMEM: No SD card in socket
-ESHAREDBUSY: Shared SPI port busy

LIBRARY
SDFLASH.LIB

sdspi_setLED

void sdspi_setLED( sd_device * sd, char state );

DESCRIPTION
This function sets the LED for the given SD card based on state. If state is 0, the LED is turned off. If state is not zero, the LED is turned on.

PARAMETER
sd Pointer to sd_device structure of the SD card.
state The state to set the LED to: 0 = Off and Non-zero = On

LIBRARY
SDFLASH.LIB
sdspi_set_block_length

```c
int sdspi_set_block_length( sd_device * sd, int block_length );
```

**DESCRIPTION**

This function executes protocol command 16 to set the block length for the SD card. The default block length for SD cards is 512 bytes. Please note that `sdspi_write_sector()` and `sdspi_read_sector()` work on 512 byte blocks only. If you change the block size, these functions will need to be modified, or you will need to execute commands directly through `sdspi_process_command()` and internal write block and read block functions.

**PARAMETER**

- **sd**
  Pointer to device structure of the SD card.

- **block_length**
  The block size in bytes for the SD card.

**RETURN VALUE**

- 0: Success
- -EIO: I/O error
- -EINVAL: Invalid parameter given
- -ENOMEDIUM: No SD card in socket
- -ESHAREDBUSY: Shared SPI port busy

**LIBRARY**

`SDFLASH.LIB`
int sdspi_WriteContinue( sd_device * sd );

DESCRIPTION
This function completes the previously started write command to the SD card when non-blocking mode is enabled. It looks for the end of the busy signal from the card, then strobes the chip select. This function should be called repeatedly until the -EBUSY code is not returned, at which point the SPI port is freed. There is a timeout mechanism for the busy signal. If exceeded, the port is freed and the -EIO error code is returned.

PARAMETERS
sd
The device structure for the SD card.

RETURN VALUE
0: Success
-EIO: I/O error or timeout
-EBUSY: SD card is busy with write operation; call sdspi_WriteContinue() again

LIBRARY
SDFLASH.LIB
sdspi_write_sector

int sdspi_write_sector( sd_device * sd, unsigned long sector_number, char * data_buffer );

DESCRIPTION

This function is called to execute protocol command 24 to write a 512 byte block of data to the SD card.

PARAMETER

sd Pointer to device structure of the SD card.

sector_number The sector number to write.

data_buffer Pointer to a buffer of 512 bytes to write.

RETURN VALUE

0: Success
-EIO: I/O error
-EACCES: Write protected block, no write access
-EINVAL: Invalid parameter given
-ENOMEM: No SD card in socket
-ESHAREDBUSY: Shared SPI port busy
-EBUSY: SD card is busy with write operation; call sdsdi_WriteContinue() to complete (only when SD_NON_BLOCK is defined)

LIBRARY

SDFLASH.LIB
servo_alloc_table

```c
void servo_alloc_table( int which, int entries );
```

**DESCRIPTION**

Allocate an xmem data area for servo statistics collection. This function should be called once only (for each servo) at application startup time.

**PARAMETERS**

- `which` Servo (0 or 1)
- `entries` Number of entries to allocate. Each entry is 8 bytes, and stores 4 integer values. The maximum value for this parameter is 8190.

**LIBRARY**

SERVO.LIB

**SEE ALSO**

servo_graph, servo_read_table, servo_stats_reset

servo_closedloop

```c
void servo_closedloop( int which, int reset );
```

**DESCRIPTION**

Run specified servo in closed-loop (PID) mode.

**PARAMETERS**

- `which` Servo (0 or 1).
- `reset` Whether to reset the current command list. The command list executes even while in open loop mode (although it will have no visible effect in that mode). If reset is non-zero, then the command list will be reset to empty and the motor will halt at the current position.

**LIBRARY**

SERVO.LIB

**SEE ALSO**

servo_openloop, servo_torque
void servo_disable_0( void );

DESCRIPTION
Disable drive to the first servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```
#define SERVO_ENABLE_PORT_0        PGDR
#define SERVO_ENABLE_PORTSHADOW_0  PGDRShadow
#define SERVO_ENABLE_PIN_0         6
```

and, optionally,

```
#define SERVO_ENABLE_DDR_0         PGDDR
#define SERVO_ENABLE_DDRSHADOW_0   PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_0
```

This function is limited to toggling the output pin. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```
#define SERVO_DISABLE_0  yyyy
```

where yyyy is the name of your own function (which is assumed to take no parameters and have no return value)

LIBRARY
SERVO.LIB

SEE ALSO
servo_enable_0
void servo_disable_1( void );

DESCRIPTION

Disable drive to the second servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```c
#define SERVO_ENABLE_PORT_1        PGDR
#define SERVO_ENABLE_PORTSHADOW_1  PGDRShadow
#define SERVO_ENABLE_PIN_1         7
```

and, optionally,

```c
#define SERVO_ENABLE_DDR_1         PGDDR
#define SERVO_ENABLE_DDRSHADOW_1   PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_1
```

This function is limited to toggling the output pin. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```c
#define SERVO_DISABLE_1  yyyy
```

where `yyyy` is the name of your own function (which is assumed to take no parameters and have no return value)

LIBRARY

SERVO.LIB

SEE ALSO

servo_enable_1
void servo_enable_0( void );

DESCRIPTION
Enable drive to the first servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

#define SERVO_ENABLE_PORT_0 PGDR
#define SERVO_ENABLE_PORTSHADOW_0 PGDRShadow
#define SERVO_ENABLE_PIN_0 6

and, optionally,

#define SERVO_ENABLE_DDR_0 PGDDR
#define SERVO_ENABLE_DDRSHADOW_0 PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_0

This function is limited to toggling the output pin high or low. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

#define SERVO_ENABLE_0 xxxx

where xxxx is the name of your own function (which is assumed to take no parameters and have no return value).

LIBRARY
SERVO.LIB

SEE ALSO
servo_disable_0
**servo_enable_1**

```c
void servo_enable_1( void );
```

**DESCRIPTION**

Enable drive to the second servo motor. This function only works if an auxiliary control signal is connected to the motor driver. The I/O pin used for this function is specified by the macros:

```c
#define SERVO_ENABLE_PORT_1 PGDR
#define SERVO_ENABLE_PORTSHADOW_1 PGDRShadow
#define SERVO_ENABLE_PIN_1 7
```

and, optionally,

```c
#define SERVO_ENABLE_DDR_1 PGDDR
#define SERVO_ENABLE_DDRSHADOW_1 PGDDRShadow
#define SERVO_ENABLE_ACTIVEHIGH_1
```

This function is limited to toggling the output pin high or low. If enabling or disabling the servo motor requires more complicated actions, you can substitute your own function by defining

```c
#define SERVO_ENABLE_1  xxxx
```

where `xxxx` is the name of your own function (which is assumed to take no parameters and have no return value).

**LIBRARY**

SERVO.LIB

**SEE ALSO**

`servo_disable_1`
void servo_gear( int count0, int count1, int slave0, int slave1 );

DESCRIPTION

NOTE: this function is currently not efficient enough for production use (owing to use of long multiplication and division). It is provided as an example of the use of callbacks from the ISR.

If two servos are in use, couple or cross-couple their positioning. This only works if NUM_SERVOS is 2, and both servos are in closed loop mode.

There are four possible sub-modes of operation, which depend on the slave0/1 parameters.

<table>
<thead>
<tr>
<th>slave0</th>
<th>slave1</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Non-gear mode: neither servo is slaved. This is the normal, default, mode.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Second servo is slaved from first servo. For every 'count0' increments of the first servo's encoder, the second servo will be moved 'count1' increments.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>First servo is slaved from second servo. For every 'count1' increments of the second servo's encoder, the first servo will be moved 'count0' increments.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Both servos cross-coupled. Movement will only result from an externally applied torque. This is a true simulation of mechanical gearing.</td>
</tr>
</tbody>
</table>

Call this function with count0 or count1 zero, or both slave0 and slave1 zero, to exit from gear mode. When a servo that was slaved is set to normal mode, its velocity is set to zero.

PARAMETERS

count0 Encoder increment for the first servo which results from count1 increments of the second servo.

count1 Encoder increment for the second servo which results from count0 increments of the first servo.

Together, count0 and count1 determine the gearing ratio. Neither value should be set to a magnitude greater than about 500, to avoid internal arithmetic overflow. In any gear mode, the total movement of either servo should be limited to less than about 2M counts in either direction from the point at which gear mode was set. If a smaller range of movement is acceptable, then the maximum of either count parameter may be increased proportionally. The value of count0/count1 or count1/count0 should not have a magnitude greater than about 10 to avoid encoder quantization problems, especially in cross-coupled mode.
servo_gear (cont’d)

slave0  1 if first servo slaved to second, else zero.
slave1  1 if second servo slaved to first, else zero.

LIBRARY
   SERVO.LIB

SEE ALSO
   servo_closedloop, servo_torque
int servo_graph( int which, word start, word nlines, word samples, word what, int low, int high );

DESCRIPTION

Draw ASCII-art graph of servo response. This is primarily intended for debugging. It should be called after resetting the sample collection table using servo_stats_reset(), then executing a movement whose response is to be graphed.

PARAMETERS

- **which**: Servo (0 or 1)
- **start**: Starting sample number
- **nlines**: Number of lines (sample bins) in graph - vertical axis
- **samples**: Number of samples to cover (should be multiple of nlines)
- **what**: Which statistic to print: 0 is for error; 1 for error integral; 2 for error rate (differential), 3 for PWM output setting. These may be customized to have different meanings
- **low**: Low range of horizontal axis
- **high**: High range of horizontal axis

RETURN VALUE

- 0: OK
- -1: error

LIBRARY

SERVO.LIB

SEE ALSO

servo_alloc_table, servo_read_table, servo_stats_reset
servo_init

void servo_init( void );

DESCRIPTION
This function must be called once at the beginning of application code to initialize the servo library.

LIBRARY
SERVO.LIB

SEE ALSO
servo_stats_reset, servo_alloc_table, servo_set_coeffs, servo_enable_0

servo_millirpm2vcmd

long servo_millirpm2vcmd( int which, long millirpm );

DESCRIPTION
Convert 1/1000 RPM units to velocity command value. Basic formula is:

\[
\text{vcmd} = \frac{\text{SERVO_COUNT_PER_REV}_n \cdot \text{millirpm} \cdot 65536}{60000 \cdot \text{SERVO_LOOP_RATE_HZ}}
\]

Floating point is used to retain 24 bit precision.

PARAMETERS

which 
Servo (0 or 1).

millirpm 
Input in units of 1/1000 RPM.

RETURN VALUE
Output in units suitable for command velocity setting i.e units of 1/65536 encoder counts per ISR execution (sample).

LIBRARY
SERVO.LIB

SEE ALSO
servo_move_to, servo_set_vel, servo_set_pos
servo_move_to

```c
int servo_move_to( int which, long pos, long ticks, long accel_ticks, long final_v );
```

DESCRIPTION

Move to new position, pos. Assumes current position is “cmd” and current velocity is “vcmd” (with the values of these read from the control structure at beginning of routine).

Each "tick" represents the time interval between loop updates. This routine measures time intervals in units of ticks.

`accel_ticks` (<= `ticks`) is the number of ticks allocated to acceleration/deceleration phase of movement. The remaining part of the movement is performed at constant velocity. Acceleration and deceleration are computed to be of the same magnitude at beginning and end of motion (but may be opposite signs). `final_v` is the velocity to be achieved at end of movement. This routine returns as soon as the necessary command list is installed for execution by the ISR. The movement will not be completed until “ticks” ISR executions.

NB: if the average velocity (vt) required to complete the movement is greater than +/-16k counts per tick, then the movement is stretched to a longer time interval so as to make the peak velocity equal to the +/- 8k counts/tick (which is higher than any physical motor can follow). `accel_ticks` is set to 16384 if it is over that (since rounding errors can accumulate over long periods of low acceleration).

If this routine is called again before the previous motion is completed, then the previous motion will be overridden by the new motion. This routine uses floating point, since the mathematics are quite complex. It takes several milliseconds to execute, so should not be called to perform motions which complete in less than, say, 50ms.

This routine does not attempt to control rate of change of acceleration ("jerk" or d^3x/dt^3). It approximates the required movement profile as parabolic (constant acceleration) and linear (constant velocity) segments.

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>which</code></td>
<td>Servo (0 or 1).</td>
</tr>
<tr>
<td><code>pos</code></td>
<td>Position to be achieved at end of movement.</td>
</tr>
<tr>
<td><code>ticks</code></td>
<td>Number of ISR executions (loop update rate) over which to complete the movement. If less than 1, it is set to 1.</td>
</tr>
<tr>
<td><code>accel_ticks</code></td>
<td>Number of ticks over which acceleration is to be applied. The remainder of the interval, <code>ticks - accel_ticks</code>, is performed at constant velocity. If greater than &quot;ticks&quot;, it is set equal to &quot;ticks&quot;.</td>
</tr>
<tr>
<td><code>final_v</code></td>
<td>Final velocity to be achieved at end of movement.</td>
</tr>
</tbody>
</table>
**servo_move_to (cont’d)**

RETURN VALUE

0: OK.
1: computed velocity is "extremely high": time interval stretched to make velocity fit within allowalbe fixed-point limits (i.e. 8192 encoder counts per sample).

LIBRARY

SERVO.LIB

SEE ALSO

servo_set_vel, servo_set_pos, servo_millirpm2vcmd

---

**servo_openloop**

```c
void servo_openloop( int which, word pwm );
```

**DESCRIPTION**

Run specified servo in open-loop mode (no PID control). Note that this bypasses dynamic current-limiting (if any defined) so should be used with caution.

**PARAMETERS**

- `which` Servo (0 or 1).
- `pwm` Output PWM setting (0-1024). 0 indicates maximum reverse speed, 1024 is maximum forward speed. 512 is nominally zero speed (but this depends on amplifier offset).

LIBRARY

SERVO.LIB

SEE ALSO

servo_closedloop, servo_torque
servo_qd_zero_0

```c
void servo_qd_zero_0( void );
```

**DESCRIPTION**

Reset the first servo encoder reading to zero. The servo motor is not moved; only the notion of the current position is reset to zero. This should only be called when the servo is in open loop mode.

**LIBRARY**

SERVO.LIB

**SEE ALSO**

servo_qd_zero_1

servo_qd_zero_1

```c
void servo_qd_zero_1 (void ;)
```

**DESCRIPTION**

Reset the second servo encoder reading to zero. The servo motor is not moved; only the notion of the current position is reset to zero. This should only be called when the servo is in open loop mode.

**LIBRARY**

SERVO.LIB

**SEE ALSO**

servo_qd_zero_0
servo_read_table

int servo_read_table(int which, word entry, word nent, int data[12]);

DESCRIPTION

Read one or more table entries, returning average, max and min of all samples in the specified
group starting at entry, for nent samples.

PARAMETERS

which Servo (0 or 1)
entry First sample number
nent Number of entries starting at "entry"
data[12] Returned data: 3 sets of 4 contiguous entries. The first set (data[0]..data[3])
contains the average; the second set (data[4]..data[7]) contains the maximum;
and the last set (data[8]..data[11]) contains the minimum. The elements of each set correspond with the table data: the first element is the
instantaneous error; the second is the error integral; the third is the error rate; and the 4th is the PWM output. These may be customized to have dif-
ferent meanings.

RETURN VALUE

0: OK
1: no such entry or entries.

LIBRARY

SERVO.LIB

SEE ALSO

servo_alloc_table, servo_graph, servo_stats_reset
**servo_set_coeffs**

```c
void servo_set_coeffs( int which, int prop, int integral, int diff );
```

**DESCRIPTION**

Set the PID closed loop control coefficients. The normal sign for all coefficients should be positive in order to implement a stable control loop. See Technical Note 233 for details.

**PARAMETERS**

- **which**: Servo (0 or 1)
- **prop**: Proportional coefficient
- **integral**: Integral ("reset") coefficient
- **diff**: Derivative ("rate") coefficient

**LIBRARY**

SERVO.LIB

**SEE ALSO**

servo_closedloop, servo_openloop
void servo_set_pos( int which, long pos, long vel );

DESCRIPTION

Move the specified servo motor to a specified position and set the specified velocity at that position. This cancels any move which is currently in effect.

PARAMETERS

- **which**
  - Servo (0 or 1)

- **pos**
  - Position, as an encoder count

- **vel**
  - Velocity, in units of encoder counts per loop update interval, times 65536. You can convert RPM to a suitable velocity command using `servo_millirpm2vcmd()`.

LIBRARY

SERVO.LIB

SEE ALSO

- `servo_move_to`, `servo_set_vel`, `servo_millirpm2vcmd`
### servo_set_vel

```c
void servo_set_vel( int which, long vel );
```

**DESCRIPTION**
Move the specified servo motor at a constant velocity. This cancels any move that is currently in effect.

**PARAMETERS**
- `which` Servo (0 or 1).
- `vel` Velocity, in units of encoder counts per loop update interval, times 65536. You can convert RPM to a suitable velocity command using `servo_millirpm2vcmd()`.

**LIBRARY**
SERVO.LIB

**SEE ALSO**
- `servo_move_to`, `servo_set_pos`, `servo_millirpm2vcmd`

### servo_stats_reset

```c
void servo_stats_reset( int which );
```

**DESCRIPTION**
Reset the statistics table. This is used immediately prior to a command movement, so that the table is filled with the results of the movement command. Once reset, one table row is filled in for each execution of the update loop (ISR driven). This continues until the table is full, or it is reset again.

**PARAMETER**
- `which` Servo (0 or 1)

**LIBRARY**
SERVO.LIB

**SEE ALSO**
- `servo_graph`, `servo_read_table`
void servo_torque(int which, int torque);

DESCRIPTION
Run specified servo in open loop controlled torque mode. The torque is limited by the dynamic current limit feature, if available.

PARAMETERS
- **which**  Servo (0 or 1)
- **torque** Amount of torque expressed as a fraction of the maximum permissible torque, times 10,000. For example, to set the torque to 1/10 the maximum value in the reverse direction, call `servo_torque(0, -1000)`.

LIBRARY
SERVO.LIB

SEE ALSO
- `servo_closedloop`, `servo_openloop`
serCheckParity

```c
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 is A-F */

DESCRIPTION

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

The functions serEclose() and serFclose() may be used with the Rabbit 3000 and Rabbit 4000.

LIBRARY

RS232.LIB

serXdatabits

void serXdatabits ( state ); /* where X is A-F */

DESCRIPTION

Sets the number of data bits in the serial format for this channel. Currently seven or eight bit modes are supported. A call to serXopen() must be made before calling this function. This function is non-reentrant.

The functions serEdatabits() and serFdatabits() may be used with the Rabbit 3000 and Rabbit 4000.

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

PARAMETERS

state An integer indicating what bit mode to use. It is best to use one of the macros provided for this:

- PARAM_7BIT - Configures serial port to use 7 bit data.
- PARAM_8BIT - Configures serial port to use 8 bit data (default condition).

LIBRARY

RS232.LIB
int serXdmaOff( void ); /* where X is A-F */

DESCRIPTION

Stops DMA transfers and unallocates the channels. Restarts the serial interrupt capability.

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

RETURN VALUE

0: Success
DMA Error codes: Error

LIBRARY

RS232.LIB

SEE ALSO

serXdmaOn
**serXdmaOn**

```c
int serXdmaOn( int tcmask, int rcmask ); /* where X is A-F */
```

**DESCRIPTION**

Enables DMA for serial send and receive. This function should be called directly after `serXopen()`.

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

**Important Flow Control Note:**
Because the DMA flowcontrol uses the external request feature, only two serial ports can use DMA flowcontrol at a time. For the CTS pin, one serial port can use PD2, PE2, or PE6, and the other can use PD3, PE3 or PE7.

**How DMA Serial Works:**

**DMA Transmit:**
When a serial function is called to transmit data, a DMA transfer begins. The length of that transfer is either the length requested, or the rest of the transmit buffer size from the current position. An interrupt is fired at the end of the transmit at which time another transmit is set up if more data is ready to go.

**DMA Receive:**
When `serXdmaOn()` is called, a continuous chain of DMA transfers begins sending any data received on the serial line to the circular buffer. With flowcontrol on, there is an interrupt after each segment of the data transfer. At that point, if receiving another segment would overwrite data, the RTSoff function is called.

For more information see the description at the beginning of RS232.LIB.

**PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tcmask</strong></td>
<td>Channel mask for DMA transmit. Use <code>DMA_CHANNEL_ANY</code> to choose any available channel.</td>
</tr>
<tr>
<td><strong>rcmask</strong></td>
<td>Channel mask for DMA receive. Use <code>DMA_CHANNEL_ANY</code> to choose any available channel.</td>
</tr>
</tbody>
</table>

**RETURN VALUE**

DMA error code or 0 for success

**LIBRARY**

RS232.LIB

**SEE ALSO**

`serXdmaOff`
void serXflowcontrolOff( void ); /* where X is A-F */

DESCRIPTION

Turns off hardware flow control for serial port X. A call to serXopen() must be made before calling this function. This function is non-reentrant.

The functions serEflowcontrolOff() and serFflowcontrolOff() may be used with the Rabbit 3000 and Rabbit 4000.

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

LIBRARY

RS232.LIB
void serXflowcontrolOn( void ); /* where X is A-F */

DESCRIPTION

Turns on hardware flow control for channel X. This enables two digital lines that handle flow control, CTS (clear to send) and RTS (ready to send). CTS is an input that will be pulled active low by the other system when it is ready to receive data. The RTS signal is an output that the system uses to indicate that it is ready to receive data; it is driven low when data can be received. A call to serXopen() must be made before calling this function.

This function is non-reentrant.

The functions serEflowcontrolOn() and serFflowcontrolOn() may be used with the Rabbit 3000 and Rabbit 4000.

If pins for the flow control lines are not explicitly defined, defaults will be used and compiler warnings will be issued. The locations of the flow control lines are specified using a set of 5 macros.

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

DESCRIPTION

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

The functions serEgetc() and serFgetc() may be used with the Rabbit 3000 and Rabbit 4000.

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

RETURN VALUE

Success: the next character in the low byte, 0 in the high byte.
Failure: -1, which indicates either an empty or a locked receive buffer.

LIBRARY

RS232.LIB

EXAMPLE

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

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

**DESCRIPTION**

Returns a byte of error flags, with bits set for any errors that occurred since the last time this function was called. Any bits set will be automatically cleared when this function is called, so a particular error will only be reported once. This function is non-reentrant.

The flags are checked with bitmasks to determine which errors occurred. Error bitmasks:

- SER_PARITY_ERROR
- SER_OVERRUN_ERROR

The functions `serEgetError()` and `serFgetError()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**RETURN VALUE**

The error flags byte.

**LIBRARY**

`RS232.LIB`
int serXopen( long baud ); /* where X is A-F */

DESCRIPTION

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

The user must define the buffer sizes for each port being used with the buffer size macros XINBUFSIZE and XOUTBUFSIZE. The values must be a power of 2 minus 1, e.g.

#define XINBUFSIZE   63
#define XOUTBUFSIZE  127

Defining the buffer sizes to $2^n - 1$ makes the circular buffer operations very efficient. If a value not equal to $2^n - 1$ is defined, a default of 31 is used and a compiler warning is given.

The functions serEopen() and serFopen() may be used with the Rabbit 3000 and Rabbit 4000.

Note: The default pin setup of Serial Port E uses parallel port C pins which conflict with the programming port. Opening serial port E with the default settings while in debug mode will therefore kill PC host/target communication.

The user must #define the following if not using the default (PCDR) settings:

SERE_TXPORT define to PEDR or PDDR
SERE_RXPORT define to PEDR or PDDR

Note: The alternate pins on parallel port D can be used for serial port B by defining SERB_USEPORTD at the beginning of a program. See the section on parallel port D in the Rabbit documentation for more detail on the alternate serial port pins.

For Rabbit 4000 Users: To use DMA for transfers, call serXdmaOn() after this function.

PARAMETERS

baud Bits per second (bps) of data transfer. Note that the baud rate must be greater than or equal to the peripheral clock frequency divided by 8192.

RETURN VALUE

1: The Rabbit's bps setting is within 5% of the input baud.
0: The Rabbit's bps setting differs by more than 5% of the input baud.

LIBRARY

RS232.LIB

SEE ALSO

serXgetc, serXpeek, serXputs, serXwrite, cof_serXgetc, cof_serXgets, cof_serXread, cof_serXputc, cof_serXputs, cof_serXwrite, serXclose
serXparity

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

**DESCRIPTION**

Sets parity mode for channel X. A call to `serXopen()` must be made before calling this function.

Parity generation for 8-bit data can be unusually slow due to the current method for generating high 9th bits. Whenever a 9th high bit is needed, the UART is disabled for approximately 10 baud times to create a long stop bit that should be recognized by the receiver as a high 9th bit.

The long delay is imposed because we are using the serial port itself to handle timing for the delay. Creating a shorter delay would require use of some other timer resource.

This function is non-reentrant.

The functions `serEparity()` and `serFparity()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**PARAMETERS**

- **parity_mode** An integer indicating what parity mode to use. It is best to use one of the macros provided:
  - `PARAM_NOPARITY` - Disables parity handling (default).
  - `PARAM_OPARITY` - Odd parity; parity bit set to “0” if odd number of 1’s in data bits.
  - `PARAM_EPARITY` - Even parity; parity bit set to “1” if even number of 1’s in data bits.
  - `PARAM_MPARITY` - Mark parity; parity bit always set to logical 1. (Rabbit 4000 only)
  - `PARAM_SPARITY` - Space parity; parity bit always set to logical 0. (Rabbit 4000 only)
  - `PARAM_2STOP` - 2 stop bits.

From a logical standpoint, the first three of these `PARAM_` macros cannot be combined, but even `PARAM_2STOP` must stand alone due to limitations in the UART hardware that will not allow parity bits and extra stop bits.

**LIBRARY**

`RS232.LIB`
**int serXpeek( void ); /* where X is A-F */**

**DESCRIPTION**

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

The functions `serEpeek()` and `serFpeek()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**RETURN VALUE**

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

**LIBRARY**

`RS232.LIB`
int serXputc( char c ); /* where X is A-F */

DESCRIPTION

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

The functions serEputc() and serFputc() may be used with the Rabbit 3000 and Rabbit 4000.

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

PARAMETERS

c Character to write to serial port X write buffer.

RETURN VALUE

0 if buffer locked or full, 1 if character sent.

LIBRARY

RS232.LIB

EXAMPLE

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

```c
int serXputs( char * s ); /* where X is A-F */
```

**DESCRIPTION**

Calls `serXwrite(s, strlen(s))`; does not write null terminator. This function is non-reentrant.

The functions `serEputs()` and `serFputs()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**PARAMETERS**

- `s` Null terminated character string to write

**RETURN VALUE**

The number of characters actually sent from serial port X.

**LIBRARY**

`RS232.LIB`

**EXAMPLE**

```c
// writes a null-terminated string of characters, repeatedly
main() {
    const static char s[] = "Hello Rabbit";
    serAopen(19200);
    while (1) {
        serAputs(s);
    }
    serAclose();
}
```
serXrdFlush

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

**DESCRIPTION**

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

The functions `serErdFlush()` and `serFrdFlush()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**LIBRARY**

`RS232.LIB`

---

serXrdFree

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

**DESCRIPTION**

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

The functions `serErdFree()` and `serFrdFree()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**RETURN VALUE**

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

**LIBRARY**

`RS232.LIB`
int serXrdUsed( void ); /* where X is A-F */

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

The functions serErdUsed() and serFrdUsed() may be used with the Rabbit 3000 and Rabbit 4000.

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

RETURN VALUE
The number of characters currently in serial port X receive buffer.

LIBRARY
RS232.LIB
serXread

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

DESCRIPTION

Reads length bytes from serial port X or until tmout milliseconds transpires between bytes. The countdown of tmout does not begin until a byte has been received. A timeout occurs immediately if there are no characters to read. This function is non-reentrant.

The functions serEread() and serFread() may be used with the Rabbit 3000 and Rabbit 4000.

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

PARAMETERS

- **data**: Data structure to read from serial port X
- **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**

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

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

The functions `serEwrFlush()` and `serFwrFlush()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**LIBRARY**
`RS232.LIB`

---

**serXwrFree**

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

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

The functions `serEwrFree()` and `serFwrFree()` may be used with the Rabbit 3000 and Rabbit 4000.

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

**RETURN VALUE**
The number of characters the serial port transmit buffer can accept before becoming full.

**LIBRARY**
`RS232.LIB`
int serXwrite( void * data, int length ); /* X is A-F */

DESCRIPTION

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

The functions serEwrite() and serFwrite() may be used with the Rabbit 3000 and Rabbit 4000.

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

PARAMETERS

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 Rabbit";
   serAopen(19200);
   while (1) {
      serAwrite(s, strlen(s));
   }
   serAclose();
}
int serXwrUsed( void ); /* where X is A-F */

DESCRIPTION

Returns the number of characters in the output buffer. This function is non-reentrant.

The functions serErdUsed() and serFrdUsed() may be used with the Rabbit 3000 and Rabbit 4000.

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

RETURN VALUE

The number of characters currently in the output buffer.

LIBRARY

RS232.LIB
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

---

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
void set32kHzDivider( int setting );

DESCRIPTION
Sets the expanded 32kHz oscillator divider for the Rabbit 3000 processor. This function does not enable running the 32kHz oscillator instead of the main clock. This function will affect the actual rate used by the processor when the 32kHz oscillator has been enabled to run by a call to use32kHzOsc().

This function is not task reentrant.

PARAMETER

setting 32kHz divider setting. The following are valid:
• OSC32DIV_1 - don't divide 32kHz oscillator
• OSC32DIV_2 - divide 32kHz oscillator by two
• OSC32DIV_4 - divide 32kHz oscillator by four
• OSC32DIV_8 - divide 32kHz oscillator by eight
• OSC32DIV_16 - divide 32kHz oscillator by sixteen

LIBRARY
SYS.LIB

SEE ALSO
useClockDivider, useClockDivider3000, useMainOsc, use32kHzOsc
**setClockModulation**

```c
void setClockModulation( int setting );
```

**DESCRIPTION**

Changes the setting of the Rabbit 3000 CPU clock modulation. Calling this function will force a 500 clock delay before the setting is changed to ensure that the previous modulation setting has cleared before the next one is set. See the *Rabbit 3000 Microprocessor User's Manual* for more details about clock modulation for EMI reduction.

**PARAMETER**

- **setting**
  
  Clock modulation setting. Allowed values are:
  
  - 0 = no modulation
  - 1 = weak modulation
  - 2 = strong modulation

**LIBRARY**

```c
SYS.LIB
```
```
int set_cpu_power_mode( int mode, char clkDoubler, char shortChipSelect );
```

**DESCRIPTION**

Sets operating power of the controller. Suspend serial communication and other data transmission activity prior to calling this function, which sets higher priority interrupt while switching clock frequencies.

This function is non-reentrant.

**PARAMETERS**

**mode**  
Mode operation. Use the following table values below. (The higher the value the lower the power consumption of controller.)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cclk=Pclk=MainOsc</td>
<td>Debug capable</td>
</tr>
<tr>
<td>2</td>
<td>Cclk=Pclk=MainOsc/2</td>
<td>Debug capable (19200 baud)</td>
</tr>
<tr>
<td>3</td>
<td>Cclk=Pclk=MainOsc/4</td>
<td>Debug capable (9600 baud)</td>
</tr>
<tr>
<td>4</td>
<td>Cclk=Pclk=MainOsc/6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Cclk=Pclk=MainOsc/8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Cclk=Pclk= 32.768KHz</td>
<td>Periodic Interrupt disabled, so call hitwd()</td>
</tr>
<tr>
<td>7</td>
<td>Cclk=Pclk=32KHz/2=16.384KHz</td>
<td>Periodic Interrupt disabled, so call hitwd()</td>
</tr>
<tr>
<td>8</td>
<td>Cclk=Pclk=32KHz/4 =8.192KHz</td>
<td>Periodic Interrupt disabled, so call hitwd()</td>
</tr>
<tr>
<td>9</td>
<td>Cclk=Pclk=32KHz/8=4.096KHz</td>
<td>Periodic Interrupt disabled, so call hitwd()</td>
</tr>
<tr>
<td>10</td>
<td>Cclk=Pclk=32kHz/16 =2.048KHz</td>
<td>Periodic Interrupt disabled, so call hitwd()</td>
</tr>
</tbody>
</table>
set_cpu_power_mode (cont’d)

**clkDoubler**  
Clock doubler setting: CLKDOUBLER_ON or CLKDOUBLER_OFF.  
CPU will operate at half selected speed when turned off. This parameter only affects main oscillator modes, not 32 kHZ oscillator modes. Turning Clock doubler off reduces power consumption.

**shortChipSelect**  
Short Chip Select setting. Use SHORTCS_OFF, or SHORTCS_ON.

*Note:* When short chip select is on, make sure that interrupts are disabled during I/O operations. Turning Short Chip Select on may reduce power consumption. See the Rabbit processor manual for more information regarding chip selects and low power operation.

**RETURN VALUE**
- 0: valid parameter
- -1: invalid parameter

**LIBRARY**
- low_power.lib
int setjmp( jmp_buf env );

DESCRIPTION

Store the PC (program counter), SP (stack pointer) and other information about the current state into env. The saved information can be restored by executing longjmp().

Note: you cannot use setjmp() to move out of slice statements, costatements, or cofunctions.

Typical usage:

```c
switch (setjmp(e)) {
    case 0:    // first time
        f();    // try to execute f(), may call longjmp()
        break;  // if we get here, f() was successful
    case 1:    // to get here, f() called longjmp()
        /* do exception handling */
        break;
    case 2:    // similar to above, but different exception code
        ...
}

f() {
    g()
    ...
}

g() {
    ...
    longjmp(e,2);    // exception code 2, jump to setjmp() statement,
                     // setjmp() returns 2, so execute
                     // case 2 in the switch statement
}
```

PARAMETERS

env Information about the current state

RETURN VALUE

Returns zero if it is executed. After longjmp() is executed, the program counter, stack pointer and etc. are restored to the state when setjmp() was executed the first time. However, this time setjmp() returns whatever value is specified by the longjmp() statement.

LIBRARY

SYS.LIB

SEE ALSO

longjmp
long SetSerialTATxRValues( long bps, char *divisor, int tatXr );

DESCRIPTION

Sets up the possibly shared serial timer (TATxR) resources required to achieve, as closely as possible, the requested serial bps rate. The algorithm attempts to find, when necessary and if possible, the lowest value for the TAT1R that will precisely produce the requested serial bps rate. For this reason, an application that requires the TAT1R to be shared should generally first set up its usage with (1) the most critical timer A1 cascade rate, or (2) the lowest timer A1 cascade rate. That is, consider setting up the most critical stage (PWM, servo, triac, ultra-precise serial rate, etc.) first, else set up the slowest usage (often, the lowest serial rate) first.

Note that this function provides no TATxR resource sharing protection for an application that uses any of the individual TATxR resources either directly or indirectly. For example, this function affords no protection to an application that sets a direct usage TAT7R timer interrupt and also opens serial port D such that TAT7R is used to set the serial data rate.

A run time error occurs if parameter(s) are invalid. Also, this function is not reentrant.

PARAMETERS

- **bps**: The requested serial bits per second (BPS, baud) rate.
- **divisor**: An optional pointer to the caller's serial timer divisor variable. If the caller is not interested in the actual serial timer (TATxR) divisor value that is set by this function, then NULL may be passed.
- **tatXr**: The TATxR for the serial timer whose value(s) are to be set. Use exactly one of the following macros:
  - TAT4R for serial port A
  - TAT5R for serial port B
  - TAT6R for serial port C
  - TAT7R for serial port D
  - TAT2R for serial port E
  - TAT3R for serial port F

RETURN VALUE

The actual serial rate BPS (baud) setting that was achieved.

LIBRARY

sys.lib

SEE ALSO

TAT1R_SetValue
**SetVectExtern2000**

```c
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
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
unsigned SetVectExtern4000( int interruptNum, void * isr );

DESCRIPTION
Function to set one of the external interrupt jump table entries for the Rabbit 4000, Rabbit 3000 and some versions of the Rabbit 2000. All Rabbit interrupts use jump vectors. See SetVectIntern() for more information.

PARAMETERS
interruptNum External interrupt number. 0 and 1 are the only valid values.
isr ISR handler address. Must be a root address.

RETURN VALUE
Jump address in vector table.

LIBRARY
SYS.LIB

SEE ALSO
GetVectExtern3000, SetVectIntern, GetVectIntern
SetVectIntern

unsigned SetVectIntern( int vectNum, void * isr );

DESCRIPTION

Sets an internal interrupt table entry. All Rabbit interrupts use jump vectors. This function writes a `jp` instruction (0xC3) followed by the 16 bit ISR address to the appropriate location in the vector table. The location in RAM of the vector table is determined and set by the BIOS automatically at startup. The start of the table is always on a 0x100 boundary.

It is perfectly permissible to have ISRs in xmem and do long jumps to them from the vector table. It is even possible to place the entire body of the ISR in the vector table if it is 16 bytes long or less, but this function only sets up jumps to 16 bit addresses.

The following table shows the `vectNum` value for each peripheral or RST. The offset into the vector table is also shown. The following vectors are valid for all Rabbit processors.

<table>
<thead>
<tr>
<th>Peripheral or RST</th>
<th>vectNum</th>
<th>Vector Table Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic interrupt</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>RST 10 instruction</td>
<td>0x02</td>
<td>0x20</td>
</tr>
<tr>
<td>RST 38 instruction</td>
<td>0x07</td>
<td>0x70</td>
</tr>
<tr>
<td>Slave Port</td>
<td>0x08</td>
<td>0x80</td>
</tr>
<tr>
<td>Timer A</td>
<td>0x0A</td>
<td>0xA0</td>
</tr>
<tr>
<td>Timer B</td>
<td>0x0B</td>
<td>0xB0</td>
</tr>
<tr>
<td>Serial Port A</td>
<td>0x0C</td>
<td>0xC0</td>
</tr>
<tr>
<td>Serial Port B</td>
<td>0x0D</td>
<td>0xD0</td>
</tr>
<tr>
<td>Serial Port C</td>
<td>0x0E</td>
<td>0xE0</td>
</tr>
<tr>
<td>Serial Port D</td>
<td>0x0F</td>
<td>0xF0</td>
</tr>
</tbody>
</table>

The following vectors are valid starting with the Rabbit 3000.

<table>
<thead>
<tr>
<th>Peripheral or RST</th>
<th>vectNum</th>
<th>Vector Table Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Capture</td>
<td>0x1A</td>
<td>0x01A0</td>
</tr>
<tr>
<td>Quadrature Encoder</td>
<td>0x19</td>
<td>0x0190</td>
</tr>
<tr>
<td>Serial port E</td>
<td>0x1C</td>
<td>0x01C0</td>
</tr>
<tr>
<td>Serial port F</td>
<td>0x1D</td>
<td>0x01D0</td>
</tr>
</tbody>
</table>
SetVectIntern (cont’d)

The following vectors are valid starting with the Rabbit 3000 Revision 1.

<table>
<thead>
<tr>
<th>Peripheral or RST</th>
<th>vectNum</th>
<th>Vector Table Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulse Width Modulator</td>
<td>0x17</td>
<td>0x0170</td>
</tr>
<tr>
<td>Secondary Watchdog</td>
<td>0x01</td>
<td>0x10</td>
</tr>
</tbody>
</table>

The following vectors are valid starting with the Rabbit 4000.

<table>
<thead>
<tr>
<th>Peripheral or RST</th>
<th>vectNum</th>
<th>Vector Table Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer C</td>
<td>0x1F</td>
<td>0x01F0</td>
</tr>
<tr>
<td>Network Port A</td>
<td>0x1E</td>
<td>0x01E0</td>
</tr>
</tbody>
</table>

The following three RSTs are included for completeness, but should not be set by the user as they are used by Dynamic C.

<table>
<thead>
<tr>
<th>Peripheral or RST</th>
<th>vectNum</th>
<th>Vector Table Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST 18 instruction</td>
<td>0x03</td>
<td>0x30</td>
</tr>
<tr>
<td>RST 20 instruction</td>
<td>0x04</td>
<td>0x40</td>
</tr>
<tr>
<td>RST 28 instruction</td>
<td>0x05</td>
<td>0x50</td>
</tr>
</tbody>
</table>

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
    SFFLASH.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
    SFFLASH.LIB
int sf_init( void );

DESCRIPTION

Initializes serial flash chip. This function must be called before the serial flash can be used. Currently supported devices are:

- AT45DB041
- AT45DB081
- AT45DB642
- AR45DB1282

Note: This function blocks and only works on boards with one serial flash device.

RETURN VALUE

0 for success
-1 if no flash chip detected
-2 if error communicating with flash chip
-3 if unknown flash chip type

LIBRARY

SFLASH.LIB
### sf_initDevice

```c
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 `sf_device` 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 `cs_port`.

`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**

```c
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**

```c
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`
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

RETURN VALUE

0 for success
-1 for error

LIBRARY

SFLASH.LIB
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 `sf_device` 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 `sf_XWriteRAM`)
  - `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`
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 sf_device 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
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:
  - **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. Ignored on R4000 based cores, but reserved for legacy code support.
  - **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`
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 sf_device 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

```c
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 to 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

```c
int sfspi_init(void);
```

**DESCRIPTION**

Initialize SPI driver for use with serial flash. This must be called before any calls to `sf_initDevice()`.

**RETURN VALUE**

- 0 for success
- -1 for error

**LIBRARY**

`SFLASH.LIB`
**sin**

```c
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**

```c
float sinh( float x );
```

**DESCRIPTION**

Computes the hyperbolic sine of x. This functions takes a unitless number as a parameter and returns a unitless number.

**PARAMETERS**

- **x**
  
  Value to compute.

**RETURN VALUE**

The hyperbolic sine of x.

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

**LIBRARY**

MATH.LIB

**SEE ALSO**

sin, asin, cosh, tanh
**snprintf**

```c
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,

```c
snprintf(buffer, BUF_LEN, "%s=%x","variable x",256);
```

puts the string “variable x=100” into `buffer`.

A complete list of valid conversion specifiers (%d, %s, etc.) can be found in the description for `printf()` under Dynamic C Conversion Specifiers.

The macro `STDIO_DISABLE_FLOATS` can be defined if it is not necessary to format floating point numbers. If this macro is defined, %e, %f and %g will not be recognized. This can save thousands of bytes of code space.

This function can be called by processes of different priorities.

**PARAMETERS**

- `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**

```c
void SPIinit( void );
```

**DESCRIPTION**

Initialize the SPI port parameters for a serial interface only. This function does nothing for a parallel interface. A description of the values that the user may define before the `#use SPI.LIB` statement is found at the top of the library `Lib\Spi\Spi.lib`.

**LIBRARY**

`SPI.LIB`

**SEE ALSO**

`SPIRead`, `SPIWrite`, `SPIWrRd`
SPIRead

`void SPIRead( void * DestAddr, int ByteCount );`

**DESCRIPTION**

Reads a block of bytes from the SPI port. The variable `SPIxor` needs to be set to either 0x00 or 0xFF depending on whether or not the received signal needs to be inverted. Most applications will not need inversion. `SPIinit()` sets the value of `SPIxor` to 0x00.

If `SPI_SLAVE_RDY_PORT` is defined for a slave device the driver will turn on the bit immediately upon activating the receiver. It will then wait for a byte to become available then turn off the bit. The byte will not be available until the master supplies the 8 clock pulses.

If `SPI_SLAVE_RDY_PORT` is defined for a master device the driver will wait for the bit to become true before activating the receiver and then wait for it to become false after receiving the byte.

Note for Master: the receiving device Chip Select must already be active.

**PARAMETERS**

- **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
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
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**

```c
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**

```c
void srand( unsigned long seed );
```

**DESCRIPTION**
Sets the seed value for the \( \text{rand}() \) function.

**PARAMETER**
- \( seed \) This must be an odd number.

**LIBRARY**
MATH.LIB

**SEE ALSO**
- rand, randb, randg
**strcat**

NEAR SYNTAX: `char * _n_strcat( char * dst, char * src );`
FAR SYNTAX: `char far * _f_strcat( char far * dst, char far * src );`

*Note:* By default, `strcat()` is defined to `_n_strcat()`.

**DESCRIPTION**

Concatenate string `src` to the end of `dst`.

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

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

**PARAMETERS**

- `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

NEAR SYNTAX: char * _n_strchr( char * src, char ch );
FAR SYNTAX: char far * _f_strchr( char far * src, char ch );

Note: By default, strchr() is defined to _n_strchr().

DESCRIPTION

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

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

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

PARAMETERS

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**

**NEAR SYNTAX:** int _n_strcmp( char * str1, char * str2 );

**FAR SYNTAX:** int _f_strcmp( char far * str1, char far * str2 );

**Note:** By default, strcmp() is defined to _n_strcmp().

**DESCRIPTION**

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

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

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

**PARAMETERS**

- **str1** Pointer to string 1.
- **str2** Pointer to string 2.

**RETURN VALUE**

- **<0:** str1 is less than str2 because character in str1 is less than corresponding character in str2, or str1 is shorter than but otherwise identical to str2.
- **=0:** str1 is identical to str2
- **>0:** str1 is greater than str2 because character in str1 is greater than corresponding character in str2, or str2 is shorter than but otherwise identical to str1.

**LIBRARY**

STRING.LIB

**SEE ALSO**

strncmp, strcmpi, strncmpi
**strcmi**

**NEAR SYNTAX:** `int * _n_strcmi( char * str1, char * str2 );`

**FAR SYNTAX:** `int _f_strcmi( char far * str1, char far * str2 );`

**Note:** By default, `strcmi()` is defined to `_n_strcmi()`.

**DESCRIPTION**

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

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

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

**PARAMETERS**

- `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

NEAR SYNTAX: char * _n_strcpy( char * dst, char * src );
FAR SYNTAX: char far * _f_strcpy( char far * dst, char far * src );

Note: By default, string is defined to _n_strcpy().

DESCRIPTION
Copies one string into another string, including the null terminator.

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

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

PARAMETERS

dst Pointer to location to receive string.

src Pointer to location to supply string.

RETURN VALUE
Pointer to destination string.

LIBRARY
STRING.LIB

SEE ALSO
strncpy
**strcspn**

**NEAR SYNTAX:**

```c
unsigned int _n_strcspn( char * s1, char * s2 );
```

**FAR SYNTAX:**

```c
size_t _f_strcspn( char far * s1, char far * s2 );
```

*Note:* By default, `strcspn()` is defined to `_n_strcspn()`.

**DESCRIPTION**

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

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

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

**PARAMETERS**

- `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

NEAR SYNTAX: int _n_strlen( char * s );
FAR SYNTAX: int _f_strlen( char far * s );

Note: By default, strlen() is defined to _n_strlen().

DESCRIPTION
Calculate the length of a string.

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

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the USE_FAR_STRING macro is defined and all pointers are near pointers, append _n_ to the function name, e.g., _n_strfunc.

For more information about FAR pointers, see the Dynamic C User’s Manual or the samples in Samples/Rabbit4000/FAR/.

PARAMETERS
s Character string.

RETURN VALUE
Number of bytes in a string.

LIBRARY
STRING.LIB
strncat

NEAR SYNTAX: char * _n_strncat( char *dst, char *src, unsigned int n );
FAR SYNTAX: char far * _f_strncat( char far * dst, char far * src, size_t n );

Note: By default, strncat() is defined to _n_strncat().

DESCRIPTION

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

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

Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the USE_FAR_STRING macro is defined and all pointers are near pointers, append _n_ to the function name, e.g., _n_strfunc.

For more information about FAR pointers, see the Dynamic C User’s Manual or the samples in Samples/Rabbit4000/FAR/.

PARAMETERS

<table>
<thead>
<tr>
<th>dst</th>
<th>Pointer to location to receive string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Pointer to location to supply string.</td>
</tr>
<tr>
<td>n</td>
<td>Maximum number of bytes to copy. If equal to zero, this function has no effect.</td>
</tr>
</tbody>
</table>

RETURN VALUE

Pointer to destination string.

LIBRARY

STRING.LIB

SEE ALSO

strcat
**strncmp**

**NEAR SYNTAX:** int _n_strncmp( char * str1, char * str2, n );

**FAR SYNTAX:** int _f_strncmp( char far * str1, char far * str2, unsigned n );

**Note:** By default, strncmp() is defined to _n_strncmp().

**DESCRIPTION**

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

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

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

**PARAMETERS**

- **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**

**NEAR SYNTAX:**
```
int _n_strncmpi( char * str1, char * str2, unsigned n );
```

**FAR SYNTAX:**
```
int _f_strncmpi( char far * str1, char far * str2, unsigned n );
```

**Note:** By default, strncmpi() is defined to _n_strncmpi().

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

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

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

**PARAMETERS**
- `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**
strcapi, strcmp, strncmp
strncpy

NEAR SYNTAX: char * _n_strncpy( char *dst, char *src, unsigned int n );
FAR SYNTAX: char far * _f_strncpy( char far * dst, char far * src,
    size_t n );

Note: By default, strncpy() is defined to _n_strncpy().

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

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

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

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

NEAR SYNTAX: char * _n_strpbrk( char * s1, char * s2 );
FAR SYNTAX: char far * _f_strpbrk( char far * s1, char far * s2 );

Note: By default, strpbrk() is defined to _n_strpbrk().

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

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

PARAMETERS
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**

**NEAR SYNTAX:** char * _n_strrchr( char * s, int c );
**FAR SYNTAX:** char far * _f_strrchr( char far * s, int c );

**Note:** By default, `strrchr()` is defined to `_n_strrchr()`.

**DESCRIPTION**

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

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

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

**PARAMETERS**

- `s` String to be searched
- `c` Search character

**RETURN VALUE**

Pointer to last occurrence of `c` in `s`. If `c` is not found in `s`, return null.

**LIBRARY**

`STRING.LIB`

**SEE ALSO**

- `strchr`, `strcspn`, `strtok`
**strspn**

**NEAR SYNTAX:** size_t _n_strspn( char * src, char * brk );  
**FAR SYNTAX:** size_t _f_strspn( char far * src, char far * brk );

Note: By default, `strspn()` is defined to `n_strspn()`.

**DESCRIPTION**

Scans a string for the first segment in `src` containing only characters specified in `brk`.  
For Rabbit 4000+ users, this function supports FAR pointers. By default the near version of the function is called. The macro `USE_FAR_STRING` will change all calls to functions in this library to their far versions. The user may also explicitly call the far version with `_f_strfunc` where `strfunc` is the name of the string function.  
Because FAR addresses are larger, the far versions of this function will run slightly slower than the near version. To explicitly call the near version when the `USE_FAR_STRING` macro is defined and all pointers are near pointers, append `_n_` to the function name, e.g., `_n_strfunc`. For more information about FAR pointers, see the *Dynamic C User’s Manual* or the samples in Samples/Rabbit4000/FAR/.

**PARAMETERS**

- `src` String to be scanned  
- `brk` Set of characters

**RETURN VALUE**

Returns the length of the segment.

**LIBRARY**

STRING.LIB
**strstr**

**NEAR SYNTAX:** char * _n_strstr( char *s1, char *s2 );
**FAR SYNTAX:** char far * _f_strstr( char far * s1, char far * s2 );

*Note:* By default, strstr() is defined to _n_strstr().

**DESCRIPTION**

Finds a substring specified by s2 in string s1.

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

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

**PARAMETERS**

- **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**

**NEAR SYNTAX:** float _n_strtod( char * s, char ** tailptr );  
**FAR SYNTAX:** float _f_strtod( char far * s, char far * far * tailptr );

**Note:** By default, **strtod()** is defined to **_n_strtod()**.

**DESCRIPTION**

ANSI string to float conversion.

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

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

**Warning:** The far version of **strtod** is **not** backwards compatible with near pointers due to the use of a double pointer. The problem is that **char ** tailptr** is a 16-bit pointer pointing to another 16-bit pointer. The far version, **char far * far * tailptr**, is a 32-bit pointer pointing to a 32-bit pointer. If you pass a double near pointer as the argument to the double far pointer function, the double dereference (**tailptr**) of the double pointer will attempt to access a 32-bit address pointed to by the passed near pointer. The compiler does not know the contents of a pointer and will assume the inner pointer is a 32-bit pointer. For more information about FAR pointers, please see the **Dynamic C User's Manual**.
In the following examples:

\[
\begin{array}{ll}
[ ] = 1 \text{ byte} \\
[ ][ ][x][x] \text{ indicates a NEAR address (16 bit) upcast to FAR}
\end{array}
\]

Passing a “char far * far * ptr” as tailptr:

\[
\begin{array}{ll}
\text{ADDRESS:} & \text{DATA:} \\
[ ][ ][x][x] & [y][y][y][y] (\text{tailptr}) \\
[y][y][y][y] & [z][z][z][z] (*\text{tailptr}) \\
[z][z][z][z] & \text{[Correct contents]} (**\text{tailptr})
\end{array}
\]

Passing a 'char ** ptr' as tailptr: Note the first pointer can be upcast to FAR but the compiler doesn't know to upcast the internal pointer.

\[
\begin{array}{ll}
\text{ADDRESS:} & \text{DATA:} \\
[ ][ ][x][x] & \text{[ ][ ][y][y]} (\text{tailptr}) \\
[ ][ ][y][y] & \text{[?][?][z][z]} (*\text{tailptr}) \\
[?][?][z][z] & \text{[Incorrect contents]} (**\text{tailptr})
\end{array}
\]

**PARAMETERS**

\[
\begin{align*}
\text{s} & \quad \text{String to convert.} \\
\text{tailptr} & \quad \text{Pointer to a pointer of character. The next conversion may resume at the location specified by *tailptr.}
\end{align*}
\]

**RETURN VALUE**

The float number represented by “s.”

**LIBRARY**

\[\text{STRING.LIB}\]

**SEE ALSO**

\[\text{atof}\]
**strtok**

**NEAR SYNTAX:** char * _n_strtok( char * src, char * brk );

**FAR SYNTAX:** char far * _f_strtok( char far * src, char far * brk );

**Note:** By default, strtok() is defined to _n_strtok().

**DESCRIPTION**

Scans src for tokens separated by delimiter characters specified in brk.

First call with non-null for src. Subsequent calls with null for src continue to search tokens in the string. If a token is found (i.e., delineators found), replace the first delimiter in src with a null terminator so that src points to a proper null terminated token.

**PARAMETERS**

- **src** String to be scanned, must be in SRAM, cannot be a constant. In contrast, strings initialized when they are declared are stored in flash memory, and are treated as constants.

- **brk** Character delimiter.

**RETURN VALUE**

Pointer to a token. If no delimiter (therefore no token) is found, returns null.

**LIBRARY**

STRING.LIB

**SEE ALSO**

strchr, strrchr, strstr, strcspn
strtol

NEAR SYNTAX: long _n_strtol( char * sptr, char ** tailptr, int base );
FAR SYNTAX: long _f_strtol( char far * sptr, char far * far * tailptr, int base );

Note: By default, strtol() is defined to _n_strtol().

DESCRIPTION

ANSI string to long conversion.

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

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

Warning: The far version of strtod is not backwards compatible with near pointers due to the use of a double pointer. The problem is that char ** tailptr is a 16-bit pointer pointing to another 16-bit pointer. The far version, char far * far * tailptr, is a 32-bit pointer pointing to a 32-bit pointer. If you pass a double near pointer as the argument to the double far pointer function, the double dereference (**tailptr) of the double pointer will attempt to access a 32-bit address pointed to by the passed near pointer. The compiler does not know the contents of a pointer and will assume the inner pointer is a 32-bit pointer. For more information about FAR pointers, please see the Dynamic C User's Manual.

In the following examples:

[ ] = 1 byte
[ ][ ][x][x] indicates a NEAR address (16 bit) upcast to FAR

Passing a “char far * far * ptr” as tailptr:

ADDRESS:       DATA:
[ ][ ][x][x]   [y][y][y][y] (tailptr)
[y][y][y][y]   [z][z][z][z] (*tailptr)
[z][z][z][z]   [Correct contents] (**tailptr)
Passing a 'char ** ptr' as tailptr: Note the first pointer can be upcast to FAR but the compiler doesn't know to upcast the internal pointer.

ADDRESS: DATA:
[ ][ ][x][x] [ ][ ][y][y] (tailptr)
[ ][ ][y][y] [?][?][z][z] (*tailptr)
[?][?][z][z] [Incorrect contents] (**tailptr)

PARAMETERS

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( void );

DESCRIPTION
This function should be called at the start of a program if you are using protected variables. It determines whether this restart of the board is due to a software reset from Dynamic C or a call to forceSoftReset(). If it was a soft reset, this function then does the following:

• Calls _prot_init() to initialize the protected variable mechanisms. It is up to the user to initialize protected variables.
• Calls sysResetChain(). The user may attach functions to this chain to perform additional startup actions (for example, initializing protected variables). If a soft reset did not take place, this function calls _prot_recover() to recover any protected variables.

LIBRARY
SYS.LIB

SEE ALSO
chkHardReset, chkSoftReset, chkWDTO

sysResetChain

void sysResetChain ( void );

DESCRIPTION
This is a function chain that should be used to initialize protected variables. By default, it's empty.

LIBRARY
SYS.LIB
float tan ( float x );

DESCRIPTION
Compute the tangent of the argument.

Note: The Dynamic C functions \texttt{deg()} and \texttt{rad()} convert radians and degrees.

PARAMETERS
\texttt{x} \quad \text{Angle in radians.}

RETURN VALUE
Returns the tangent of \( x \), where \(-8 \times \pi \leq x \leq +8 \times \pi\). If \( x \) is out of bounds, the function returns 0 and signals a domain error. If the value of \( x \) is too close to a multiple of 90\(^\circ\) (\( \pi/2 \)) the function returns INF and signals a range error.

LIBRARY
MATH.LIB

SEE ALSO
\begin{itemize}
\item atan, cos, sin, tanh
\end{itemize}
tanh

float tanh ( float x );

DESCRIPTION
Computes the hyperbolic tangent of argument. This function 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
char TAT1R_SetValue( int requestor, int value );

DESCRIPTION

If not already in use, or if in a compatible use, allocates the TAT1R resource (sets a new or keeps the current TAT1R value) as requested. Also enables or disables the requestor's timer A1 cascade bit(s) in TACR or TBCR, as appropriate. When the timer B cascade from timer A1 is disabled in TBCR the timer B "clocked by PCLK/2" is then enabled.

A run time error occurs if parameter(s) are invalid and also, this function is not reentrant.

Note: This function does not attempt to manage interrupts that are associated with timers A or B; that work is left entirely up to the application.

PARAMETERS

requestor The requestor of the TAT1R resource. Use exactly one of the following macros to specify the appropriate requestor:

• TAT1R_A1TIMER_REQ (e.g., direct use of Timer A1)
• TAT1R_A2TIMER_REQ (e.g., use by serial port E)
• TAT1R_A3TIMER_REQ (e.g., use by serial port F)
• TAT1R_A4TIMER_REQ (e.g., use by serial port A)
• TAT1R_A5TIMER_REQ (e.g., use by serial port B)
• TAT1R_A6TIMER_REQ (e.g., use by serial port C)
• TAT1R_A7TIMER_REQ (e.g., use by serial port D)
• TAT1R_BTIMER_REQ (e.g., use with PWM, servo or triac)

value Either the new TAT1R setting value (0 to 255, inclusive), or the macro TAT1R_RELEASE_REQ to release the TAT1R resource in use by the specified requestor.

RETURN VALUE

The new or current TAT1R setting. The caller should check their requested new TAT1R value against this return value. If the two values are not the same, the caller may decide the return value is acceptable after all and make another request using the previous return value. A valid release request always succeeds; in this case there is no need for the caller to check the return value.

LIBRARY

sys.lib
`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.

```c
struct tm {
    char tm_sec;  // seconds 0-59
    char tm_min;  // 0-59
    char tm_hour; // 0-23
    char tm_mday; // 1-31
    char tm_mon;  // 1-12
    char tm_year; // 80-147 (1980-2047)
    char tm_wday; // 0-6 0==Sunday
};
```

**RETURN VALUE**
- `0`: Successful.
- `-1`: Clock read failed.

**LIBRARY**
- `RTCLOCK.LIB`

**SEE ALSO**
- `mktm`, `mktime`, `tm_wr`
### tm_wr

```c
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:

```c
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.

```c
struct tm {
    char tm_sec;       // seconds 0-59
    char tm_min;       // 0-59
    char tm_hour;      // 0-23
    char tm_mday;      // 1-31
    char tm_mon;       // 1-12
    char tm_year;      // 80-147 (1980-2047)
    char tm_wday;      // 0-6 0==Sunday
};
```

**RETURN VALUE**

- 0: Success.
- -1: Failure.

**LIBRARY**

```
RTCLOCK.LIB
```

**SEE ALSO**

- `mktm`, `mktime`, `tm_rd`
tolower

```c
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

```c
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

```c
void updateTimers( void );
```

**DESCRIPTION**

Updates the values of `TICK_TIMER`, `MS_TIMER`, and `SEC_TIMER` while running off the 32 kHz oscillator. Since the periodic interrupt is disabled when running at 32 kHz, these values will not be updated unless this function is called.

**LIBRARY**

SYS.LIB

**SEE ALSO**

`useMainOsc`, `use32kHzOsc`

use32kHzOsc

```c
void use32kHzOsc( void );
```

**DESCRIPTION**

Sets the Rabbit processor to use the 32kHz real-time clock oscillator for both the CPU and peripheral clock, and shuts off the main oscillator. If this is already set, there is no effect. This mode should provide greatly reduced power consumption. Serial communications will be lost since typical baud rates cannot be made from a 32kHz clock. Also note that this function disables the periodic interrupt, so `waitfor` and related statements will not work properly (although costatements in general will still work). In addition, the values in `TICK_TIMER`, `MS_TIMER`, and `SEC_TIMER` will not be updated unless you call the function `updateTimers()` frequently in your code. In addition, you will need to call `hitwd()` periodically to hit the hardware watchdog timer since the periodic interrupt normally handles that, or disable the watchdog timer before calling this function. The watchdog can be disabled with `Disable_HW_WDT()`.

`use32kHzOsc()` is not task reentrant.

**LIBRARY**

SYS.LIB

**SEE ALSO**

`useMainOsc`, `useClockDivider`, `updateTimers`
useClockDivider

void useClockDivider( void );

DESCRIPTION

Sets the Rabbit processor to use the main oscillator divided by 8 for the CPU (but not the peripheral clock). If this is already set, there is no effect. Because the peripheral clock is not affected, serial communications should still work. This function also enables the periodic interrupt in case it was disabled by a call to use32kHzOsc().

This function is not task reentrant.

LIBRARY

SYS.LIB

SEE ALSO

useMainOsc, use32kHzOsc
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 en-
ables 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 di-
vide the main processor clock by 8. To divide the main processor clock by any of the other al-
lowable 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**

```c
void useMainOsc( void );
```

**DESCRIPTION**

Sets the Rabbit processor to use the main oscillator for both the CPU and peripheral clock. If this is already set, there is no effect. This function also enables the periodic interrupt in case it was disabled by a call to `use32kHzOsc()`, and updates the `TICK_TIMER`, `MS_TIMER`, and `SEC_TIMER` variables from the real-time clock. This function is not task reentrant.

**LIBRARY**

`SYS.LIB`

**SEE ALSO**

`use32kHzOsc`, `useClockDivider`

---

**utoa**

```c
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`
int vram2root( void * dest, int start, int length );

DESCRIPTION

This function copies data from the VBAT RAM. Tamper detection on the Rabbit 4000 erases the VBAT RAM with any attempt to enter bootstrap mode.

PARAMETERS

- **dest**: The address to which the data in the VBAT RAM will be copied.
- **start**: The start location within the VBAT RAM (0-31).
- **length**: The length of data to read from VBAT RAM. The length should be greater than 0. The parameters \( \text{length} + \text{start} \) should not exceed 32.

LIBRARY

VBAT.LIB

SEE ALSO

root2vram


```
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., 

```c
#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 < `count` <= 255

**RETURN VALUE**

Integer id number of an unused virtual watchdog timer.

**LIBRARY**

`VDRIVER.LIB`
**VdHitWd**

```c
int VdHitWd( int ndog );
```

**DESCRIPTION**

Resets virtual watchdog counter to N counts where N is the argument to the call to VdGetFreeWd() that obtained the virtual watchdog ndog. The virtual driver counts down watchdogs every 62.5 ms. If a virtual watchdog reaches 0, this is a fatal error. Once a virtual watchdog is active it should reset periodically with a call to VdHitWd() to prevent this.

If $N = 2$, `VdHitWd()` will need to be called again for virtual watchdog `ndog` within 62.5 ms. If $N = 255$, `VdHitWd()` will need to be called again for virtual watchdog `ndog` within 15.9375 seconds.

**PARAMETERS**

- `ndog` Id of virtual watchdog returned by VdGetFreeWd()

**LIBRARY**

VDRIVER.LIB

---

**VdInit**

```c
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
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

```c
// VdReleaseWd virtual watchdog example
main() { // handle for a virtual watchdog
    int wd;
    unsigned long tm; // wd activated, 9 virtual watchdogs
    tm = SEC_TIMER;
    wd = VdGetFreeWd(255); // now available. wd must be hit
    while(SEC_TIMER - tm < 60) { // at least every 15.875 seconds
        VdHitWd(wd); // let it run for a minute
    }
    VdReleaseWd(wd); // reset counter back to 255
    // now 10 virtual watchdogs available
}
```
int WriteFlash2( unsigned long flashDst, void * rootSrc, unsigned len );

DESCRIPTION

Write \textit{len} bytes from \textit{rootSrc} to physical address \textit{flashDst} on the 2nd flash device. The source must be in root. The \textit{flashDst} address plus the sum of \textit{numbytes[]} area must be within memory quadrant(s) already mapped to the second flash.

This function is not reentrant.

\textbf{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!

\textbf{Note:} This function is extremely dangerous when used with large sector flash. Don't do it.

PARAMETERS

- \texttt{flashDst} \hspace{1cm} Physical address of the flash destination
- \texttt{rootSrc} \hspace{1cm} Pointer to the root source
- \texttt{len} \hspace{1cm} Number of bytes to write

RETURN VALUE

- 0: Success.
- -1: Attempt to write non-2nd flash area, nothing written.
- -2: \texttt{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
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 rootSrc[] and numbytes[].

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
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.LIB

SEE ALSO

read_rtc
writeUserBlock

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

DESCRIPTION

Rabbit-based boards have a System ID block located on the primary flash. (See the *Rabbit Microprocessor Designer's Handbook* for more information on the System ID block.) Version 2 and later of this ID block has a pointer to a User ID block: a place intended for storing calibration constants, passwords, and other non-volatile data.

The User block is recommended for storing all non-file data. The User block is where calibration constants are stored for boards with analog I/O. Space in the User block is limited to as small as $(8K - \text{sizeof(SysIDBlock)})$ bytes, or less, if there are calibration constants.

writeUserBlock() writes a number of bytes from root memory to the User block. This block is protected from normal writes to the flash device and can only be accessed through this function or the function writeUserBlockArray().

Using this function can cause all interrupts to be disabled for as long as 20 ms while a flash sector erases, depending on the flash type. A single call can produce as many as four of these erase delays. This will cause periodic interrupts to be missed, and can cause other interrupts to be missed as well. Therefore, it is best to buffer up data to be written rather than to do many writes.

While debugging, several consecutive calls to this function can cause a loss of target serial communications. This effect can be reduced by introducing delays between the calls, lowering the baud rate, or increasing the serial time-out value in the project file.

**Note:** See the manual for your particular board for more information before overwriting any part of the User block.

**Note:** When using a board with serial bootflash (e.g., RCM4300, RCM4310), writeUserBlock() should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using µC/OS-II and _SPI_USE_UCOS_MUTEX is defined, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error ERR_SPI_MUTEX_ERROR will occur. See the description for _rcm43_InitUCOSMutex() for more information on using µC/OS-II and _SPI_USE_UCOS_MUTEX.

**Backwards Compatibility:**

If the version of the System ID block doesn't support the User ID block, or no System ID block is present, then 8K bytes starting 16K bytes from the top of the primary flash are designated the User ID block area. However, to prevent errors arising from incompatible large sector configurations, this will only work if the flash type is small sector. Rabbit Semiconductor manufactured boards with large sector flash will have valid System and User ID blocks, so this should not be a problem on Rabbit boards.

If users create boards with large sector flash, they must install System ID blocks version 2 or greater to use or modify this function.
writeUserBlock (cont’d)

PARAMETERS

addr Address offset in User block to write to.
source Pointer to source to copy data from.
numbytes Number of bytes to copy.

RETURN VALUE

0: Successful
-1: Invalid address or range

The return values below are new with Dynamic C 10.21:
-2: No valid user block found (block version 3 or later)
-3: flash writing error

The return values below are applicable only if _SPI_USE_UCOS_MUTEX is not #defined:
ETIME: (Serial flash only, time out waiting for SPI)
positive N: (Serial flash only, SPI in use by device N)

LIBRARY

IDBLOCK.LIB

SEE ALSO

readUserBlock, writeUserBlockArray
writeUserBlockArray

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

DESCRIPTION

Rabbit Semiconductor boards are released with System ID blocks located on the primary flash. Version 2 and later of this ID block has a pointer to a User block that can be used for storing calibration constants, passwords, and other non-volatile data. The User block is protected from normal write to the flash device and can only be accessed through this function or writeUserBlock().

This function writes a set of scattered data from root memory to the User block. If the data to be written are in contiguous bytes, using the function writeUserBlock() is sufficient. Use of writeUserBlockArray() is recommended when the data to be written is in noncontiguous bytes, as may be the case for something like network configuration data.

See the designer’s handbook for your Rabbit processor (e.g., the Rabbit 4000 Designer's Handbook) for more information about the System ID and User blocks.

Note: Portions of the User block may be used by the BIOS for your board to store values, e.g., calibration constants. See the manual for your particular board for more information before overwriting any part of the User block.

Note: When using a board with serial bootflash (e.g., RCM4300, RCM4310), writeUserBlockArray() should be called until it returns zero or a negative error code. A positive return value indicates that the SPI port needed by the serial flash is in use by another device. However, if using μC/OS-II and _SPI_USE_UCOS_MUTEX is #defined, then this function only needs to be called once. If the mutex times out waiting for the SPI port to free up, the run time error ERR_SPI_MUTEX_ERROR will occur. See the description for _rcm43_InitUCOSMutex() for more information on using μC/OS-II and _SPI_USE_UCOS_MUTEX.

Backwards Compatibility:

If the System ID block on the board doesn't support the User block, or no System ID block is present, then the 8K bytes starting 16K bytes from the top of the primary flash are designated User block area. This only works if the flash type is small sector. Rabbit manufactured boards with large sector flash will have valid System ID and User blocks, so is not a problem on Rabbit boards. If users create boards with large sector flash, they must install System ID blocks version 3 or greater to use this function, or modify this function.
writeUserBlockArray

PARAMETERS

    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.

The return values below are applicable only if _SPI_USE_UCOS_MUTEX is not #defined:

    -ETIME: (Serial flash only, time out waiting for SPI)
    postive N: (Serial flash only, SPI in use by device N)

LIBRARY

    IDBLOCK.LIB
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
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
long xalloc( long sz );

DESCRIPTION
Allocates the specified number of bytes in extended memory. Starting with Dynamic C version 7.04P3, the returned address is always even (word) aligned.

Starting with Dynamic C 8, if xalloc() fails, a run-time error will occur. This is a wrapper function for _xalloc(), for backwards compatibility. It is the same as _xalloc(&sz, 1, XALLOC_MAYBBB) except that the actual allocated amount is not returned since the parameter is not a pointer.

Starting with Dynamic C 9.30, xalloc() and related functions were modified so that they are now driven by the compiler origin directives.

Note: xalloc() is not thread safe since it accesses a global static structure with no locking.

PARAMETERS
sz Number of bytes to allocate. This is rounded up to the next higher even number.

RETURN VALUE
The 20-bit physical address of the allocated data: Success.
0: Failure.

Note: Starting with Dynamic C 8, a run-time exception will occur if the function fails.

LIBRARY
STACK.LIB

SEE ALSO
root2xmem, xmem2root, xavail
long _xalloc( long * sz, word align, word type );

DESCRIPTION

Allocates memory in extended memory. If _xalloc() fails, a runtime 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
This parameter is only meaningful on boards with more than one type of RAM. For example, boards with a fast RAM and a slower battery-backed RAM like the RCM3200 or RCM3300 Use one of the following values, any other value will have undefined results.

- XALLOC_ANY (0) - any type of SRAM storage allowed
- XALLOC_BB (1) - must be battery-backed program execution SRAM (a.k.a., fast RAM).
- XALLOC_NOTBB (2) - return non-BB SRAM only.
- XALLOC_MAYBBB (3) - return non-BB SRAM in preference to BB.

RETURN VALUE

The 20-bit physical address of the allocated data on success. On error, a runtime error occurs.

Note: This return value cannot be used with pointer arithmetic.

LIBRARY

STACK.LIB

EXCEPTIONS

ERR_BADXALLOC - if could not allocate requested storage, or negative size passed.
void xalloc_stats( word parm );

DESCRIPTION

Prints a table of available xalloc() regions to the Stdio window.

This function was introduced in Dynamic C version 8. It is for debugging and educational purposes. It should not be called in a production program.

PARAMETERS

parm

Prior to Dynamic C version 9.30: reserved for future use. Set to 0.

Starting with DC 9.30: this parameter is of type long. It is the address of the data structure xbreak_t and must not be 0.

LIBRARY

MEM.LIB (XMEM.LIB prior to DC 9.30)

SEE ALSO

xalloc, _xalloc, xavail, _xavail, xrelease
long xavail( long * addr_ptr );

DESCRIPTION
Returns the maximum length of memory that may be successfully obtained by an immediate call to xalloc(), and optionally allocates that amount.

This function was introduced in Dynamic C version 7.04P3.

PARAMETERS

addr_ptr Pointer to a long word in root data memory to store the address of the block. If this pointer is null, then the block is not allocated. Otherwise, the block is allocated as if by a call to xalloc().

RETURN VALUE
The size of the largest free block available. If this is zero, then *addr_ptr will not be changed.

LIBRARY
XMEM.LIB (was in STACK.LIB prior to DC 8)

SEE ALSO
xalloc, _xalloc, _xavail, xrelease, xalloc_stats
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

addr_ptr 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 _xalloc().

align Alignment of returned block, as per _xalloc().

type Type of memory, as per _xalloc().

RETURN VALUE

The size of the largest free block available. If this is zero, then *addr_ptr will not be changed.

LIBRARY

XMEM.LIB

SEE ALSO

xalloc, _xalloc, xavail, xrelease, xalloc_stats
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)
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 mem-
ory 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
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
int xmem2root( void * dest, unsigned long int src, unsigned int len );

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

PARAMETERS

<table>
<thead>
<tr>
<th>dest</th>
<th>Logical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Physical address</td>
</tr>
<tr>
<td>len</td>
<td>Numbers of bytes</td>
</tr>
</tbody>
</table>

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
int xmem2xmem( unsigned long dest, unsigned long src, unsigned len );

DESCRIPTION

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

PARAMETERS

<table>
<thead>
<tr>
<th>dest</th>
<th>Physical address of destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Physical address of source data</td>
</tr>
<tr>
<td>len</td>
<td>Length of source data in bytes</td>
</tr>
</tbody>
</table>

RETURN VALUE

0: Success.
-1: Attempt to write flash memory area, nothing written.

LIBRARY

XMEM.LIB
long xmemchr( long src, char ch, unsigned short n );

DESCRIPTION

Search for the first occurrence of character \texttt{ch} in the xmem area pointed to by \texttt{src}.

PARAMETERS

\begin{itemize}
  \item \texttt{src} \hspace{1em} \text{s} \text{mem (linear) address of the first character to search.}
  \item \texttt{ch} \hspace{1em} \text{Character to search for.}
  \item \texttt{n} \hspace{1em} \text{Maximum number of characters to search.}
\end{itemize}

RETURN VALUE

\begin{itemize}
  \item \texttt{0}: Character was not found within \texttt{n} bytes from the start.
  \item \texttt{>0}: Physical address of the first character that matched \texttt{ch}.
\end{itemize}

LIBRARY

\texttt{XMEM.LIB}
int xmemcmp( long xstr, char * str, unsigned short n );

DESCRIPTION
Test whether xmem string at \textit{xstr} matches the root memory string at \textit{str}. \textit{n} bytes are compared.

PARAMETERS
\begin{itemize}
\item \texttt{xstr} xmem (linear) address of the first character of the first string to compare.
\item \texttt{str} root address of the first character of the second string to compare.
\item \texttt{n} Length of each string. If \textit{n} is zero, returns zero. \textit{n} must be less than or equal 4097.
\end{itemize}

RETURN VALUE
\begin{itemize}
\item 0: Exact match.
\item >0: \texttt{xstr} > \texttt{str}
\item <0: \texttt{xstr} < \texttt{str}
\end{itemize}

LIBRARY
\texttt{XMEM.LIB}
void xrelease( long addr, long sz );

DESCRIPTION
Release a block of memory previously obtained by \texttt{xalloc()} or by \texttt{xavail()} with a non-null parameter. \texttt{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 \texttt{xalloc()}/\texttt{xrelease()} must be nested in first-allocated/last-released order, similar to the execution stack. The \texttt{addr} parameter must be the return value from \texttt{xalloc()}. If not, then a run-time exception will occur. The \texttt{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 \texttt{\_xalloc()} rather than \texttt{xalloc()}. For this reason, it is recommended that your application consistently uses \texttt{\_xalloc()} rather than \texttt{xalloc()} if you intend to use this function.

PARAMETERS
\begin{itemize}
  \item \texttt{addr} Address of storage previously obtained by \texttt{\_xalloc()}.  
  \item \texttt{sz} Size of storage previously returned by \texttt{\_xalloc()}.  
\end{itemize}

LIBRARY
\texttt{XMEM.LIB}

SEE ALSO
\texttt{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 xmem (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 xmem (linear) address of the float value to set.
val value to store into the above location.

RETURN VALUE
None

LIBRARY
XMEM.LIB
xsetlong

```c
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`: `xmem` (linear) address of the long integer value to set.
- `val`: value to store into the above location.

**RETURN VALUE**
None

**LIBRARY**
`XMEM.LIB`

xstrlen

```c
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 '@':
  ```c
  xstrlen(paddr(my_str) | (long)'@' << 24);
  ```

**RETURN VALUE**
Length of string, not counting the terminator.

**LIBRARY**
`XMEM.LIB`
IMPORTANT-READ CAREFULLY: BY INSTALLING, COPYING OR OTHERWISE USING THE
ENCLOSED RABBIT DYNAMIC C SOFTWARE, WHICH INCLUDES COMPUTER SOFTWARE
("SOFTWARE") AND MAY INCLUDE ASSOCIATED MEDIA, PRINTED MATERIALS, AND
"ONLINE" OR ELECTRONIC DOCUMENTATION ("DOCUMENTATION"), YOU (ON BEHALF OF
YOURSELF OR AS AN AUTHORIZED REPRESENTATIVE ON BEHALF OF AN ENTITY) AGREE
TO ALL THE TERMS OF THIS END USER LICENSE AGREEMENT ("LICENSE") REGARDING
YOUR USE OF THE SOFTWARE. IF YOU DO NOT AGREE WITH ALL OF THE TERMS OF THIS
LICENSE, DO NOT INSTALL, COPY OR OTHERWISE USE THE SOFTWARE AND IMMEDI-
ATELY CONTACT RABBIT FOR RETURN OF THE SOFTWARE AND A REFUND OF THE PUR-
CHASE PRICE FOR THE SOFTWARE.

We are sorry about the formality of the language below, which our lawyers tell us we need to include to
protect our legal rights. If You have any questions, write or call Rabbit at (530) 757-4616, 2900 Spafford
Street, Davis, California 95616.

1. Definitions. In addition to the definitions stated in the first paragraph of this document, capitalized
words used in this License shall have the following meanings:

1.1 "Qualified Applications" means an application program developed using the Software and that
links with the development libraries of the Software.

1.1.1 "Qualified Applications" is amended to include application programs developed using the Soft-
tools WinIDE program for Rabbit processors available from Softools, Inc.

1.1.2 The MicroC/OS-II (µC/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) manufac-
tured by, for or under license from Rabbit, or (ii) based on the Rabbit 2000 microprocessor, the
Rabbit 3000 microprocessor, the Rabbit 4000 microprocessor, or any other Rabbit microproces-
sor. Qualified Systems may not be (a) designed or intended to be re-programmable by your cus-
tomer using the Software, or (b) competitive with Rabbit products, except as otherwise stated in a
written agreement between Rabbit and the system manufacturer. Such written agreement may
require an end user to pay run time royalties to Rabbit.
2. **License.** Rabbit grants to You a nonexclusive, nontransferable license to (i) use and reproduce the Software, solely for internal purposes and only for the number of users for which You have purchased licenses for (the "Users") and not for redistribution or resale; (ii) use and reproduce the Software solely to develop the Qualified Applications; and (iii) use, reproduce and distribute, the Qualified Applications, in object code only, to end users solely for use on Qualified Systems; provided, however, any agreement entered into between You and such end users with respect to a Qualified Application is no less protective of Rabbit’s intellectual property rights than the terms and conditions of this License. (iv) use and distribute with Qualified Applications and Qualified Systems the program files distributed with Dynamic C named RFU.EXE, PILOT.BIN, and COLDLOAD.BIN in their unaltered forms.

3. **Restrictions.** Except as otherwise stated, You may not, nor permit anyone else to, decompile, reverse engineer, disassemble or otherwise attempt to reconstruct or discover the source code of the Software, alter, merge, modify, translate, adapt in any way, prepare any derivative work based upon the Software, rent, lease network, loan, distribute or otherwise transfer the Software or any copy thereof. You shall not make copies of the copyrighted Software and/or documentation without the prior written permission of Rabbit; provided that, You may make one (1) hard copy of such documentation for each User and a reasonable number of back-up copies for Your own archival purposes. You may not use copies of the Software as part of a benchmark or comparison test against other similar products in order to produce results strictly for purposes of comparison. The Software contains copyrighted material, trade secrets and other proprietary material of Rabbit and/or its licensors and You must reproduce, on each copy of the Software, all copyright notices and any other proprietary legends that appear on or in the original copy of the Software. Except for the limited license granted above, Rabbit retains all right, title and interest in and to all intellectual property rights embodied in the Software, including but not limited to, patents, copyrights and trade secrets.

4. **Export Law Assurances.** You agree and certify that neither the Software nor any other technical data received from Rabbit, nor the direct product thereof, will be exported outside the United States or re-exported except as authorized and as permitted by the laws and regulations of the United States and/or the laws and regulations of the jurisdiction, (if other than the United States) in which You rightfully obtained the Software. The Software may not be exported to any of the following countries: Cuba, Iran, Iraq, Libya, North Korea, Sudan, or Syria.

5. **Government End Users.** If You are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees: (i) if the Software is supplied to the Department of Defense ("DOD"), the Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and (ii) if the Software is supplied to any unit or agency of the United States Government other than DOD, the Government's rights in the Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.
6. **Disclaimer of Warranty.** You expressly acknowledge and agree that the use of the Software and its documentation is at Your sole risk. THE SOFTWARE, DOCUMENTATION, AND TECHNICAL SUPPORT ARE PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND. Information regarding any third party services included in this package is provided as a convenience only, without any warranty by Rabbit, and will be governed solely by the terms agreed upon between You and the third party providing such services. RABBIT AND ITS LICENSORS EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. RABBIT DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. FURTHERMORE, RABBIT DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY RABBIT OR ITS AUTHORIZED REPRESENTATIVES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

7. **Limitation of Liability.** YOU AGREE THAT UNDER NO CIRCUMSTANCES, INCLUDING NEGLIGENCE, SHALL RABBIT BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE AND/OR INABILITY TO USE THE SOFTWARE, EVEN IF RABBIT OR ITS AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT SHALL RABBIT’S TOTAL LIABILITY TO YOU FOR ALL DAMAGES, LOSSES, AND CAUSES OF ACTION (WHETHER IN CONTRACT, TORT, INCLUDING NEGLIGENCE, OR OTHERWISE) EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE.

8. **Termination.** This License is effective for the duration of the copyright in the Software unless terminated. You may terminate this License at any time by destroying all copies of the Software and its documentation. This License will terminate immediately without notice from Rabbit if You fail to comply with any provision of this License. Upon termination, You must destroy all copies of the Software and its documentation. Except for Section 2 ("License"), all Sections of this Agreement shall survive any expiration or termination of this License.
9. **General Provisions.** No delay or failure to take action under this License will constitute a waiver unless expressly waived in writing, signed by a duly authorized representative of Rabbit, and no single waiver will constitute a continuing or subsequent waiver. This License may not be assigned, sublicensed or otherwise transferred by You, by operation of law or otherwise, without Rabbit’s prior written consent. This License shall be governed by and construed in accordance with the laws of the United States and the State of California, exclusive of the conflicts of laws principles. The United Nations Convention on Contracts for the International Sale of Goods shall not apply to this License. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to affect the intent of the parties, and the remainder of this License shall continue in full force and effect. This License constitutes the entire agreement between the parties with respect to the use of the Software and its documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. There shall be no contract for purchase or sale of the Software except upon the terms and conditions specified herein. Any additional or different terms or conditions proposed by You or contained in any purchase order are hereby rejected and shall be of no force and effect unless expressly agreed to in writing by Rabbit. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Rabbit.